Networking

Betriebssystemarchitektur SS 2005

Dipl.-Inf. Bernhard Rabe
Betriebssysteme & Middleware

Inhalt

- ISO/OSI Modell
- Berkeley Sockets
- Socket Programmierung

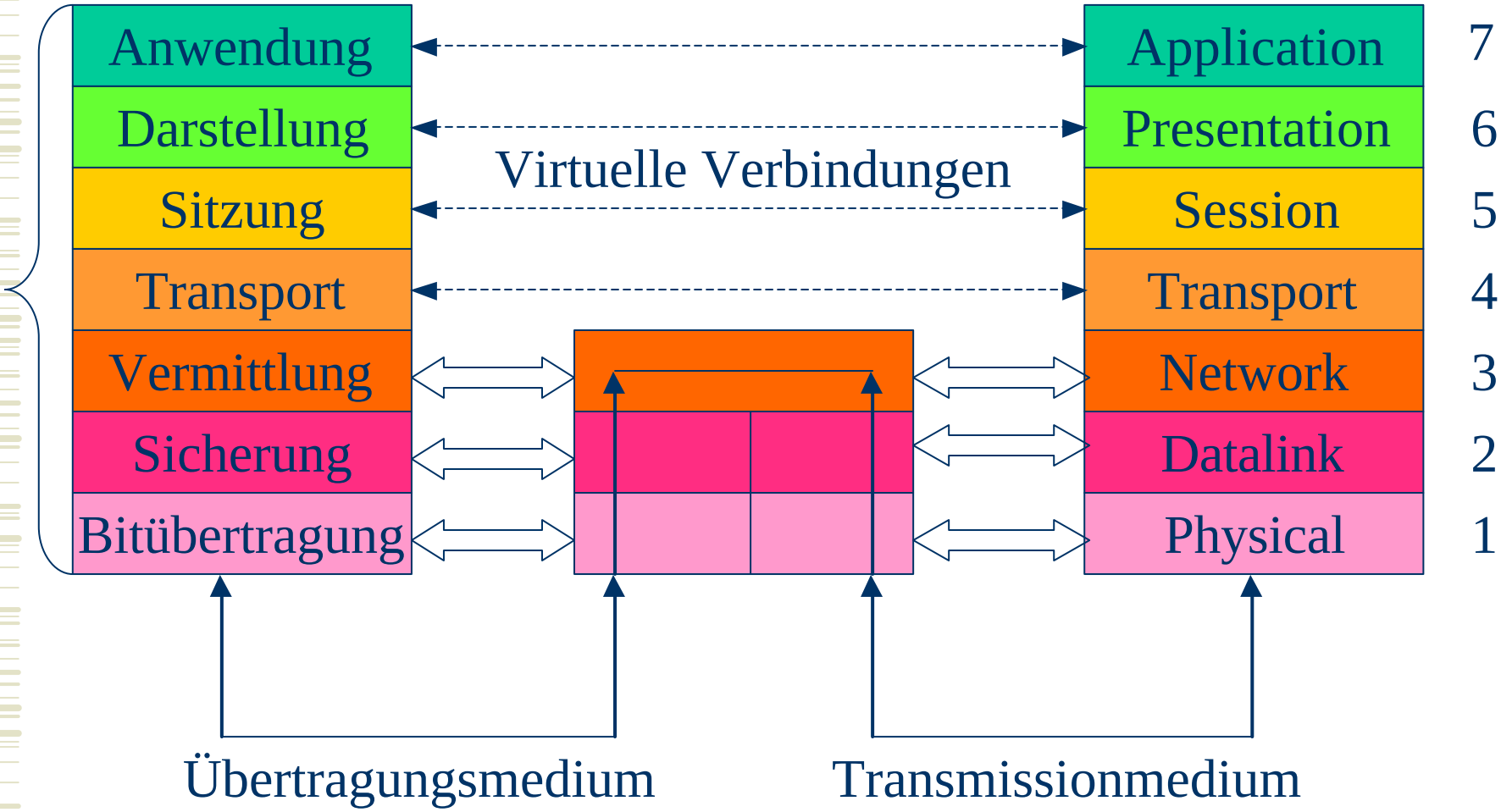
Neztwerkprogrammierung

- Rechnerübgreifende Kommunikation
  - Interprozesskommunikation
- Betriebssystemübergreifende Kommunikation
  - HTTP, FTP, SMB, NFS, ...
- Basistechnologie für unterschiedlichste Protokolle: z.B. TCP/IP, UDP/IP, IPX

ISO/OSI Referenzmodell

- Vereinheitlichen und Verbinden von Netzwerk-Software
  - **I**nternational **S**tandards **O**rganisation entwickelt ein Software Modell für die Übertragung von Nachrichten zwischen Rechern (1979-)
- **O**pen **S**ystems **I**nterconnection Referenz Modell
  - idealisiertes Kommunikationsmodell
  - jede Schicht stellt Dienste für den darrüberliegenden Schich zur Verfügung und nutzt den darunter liegende Schicht
  - jede Schicht auf einem Rechner nimmt an mit der gleichen Schicht auf dem entfernten Rechner zu kommunizieren

ISO/OSI Schichtenmodell



OSI Schichten

- **Application Layer (*Anwendungsschicht*):**
  - Datenaustausch zwischen Netzwerkanwendungen
- **Presentation Layer (*Darstellungsschicht*):**
  - Wahl einer gemeinsamen Syntax für den Datenaustausch
- **Session Layer (*Sitzungs- / Kommunikationssteuerungsschicht*):**
  - verwaltet Verbindungen (Sessions) zwischen kooperierenden Anwendungen

OSI Schichten

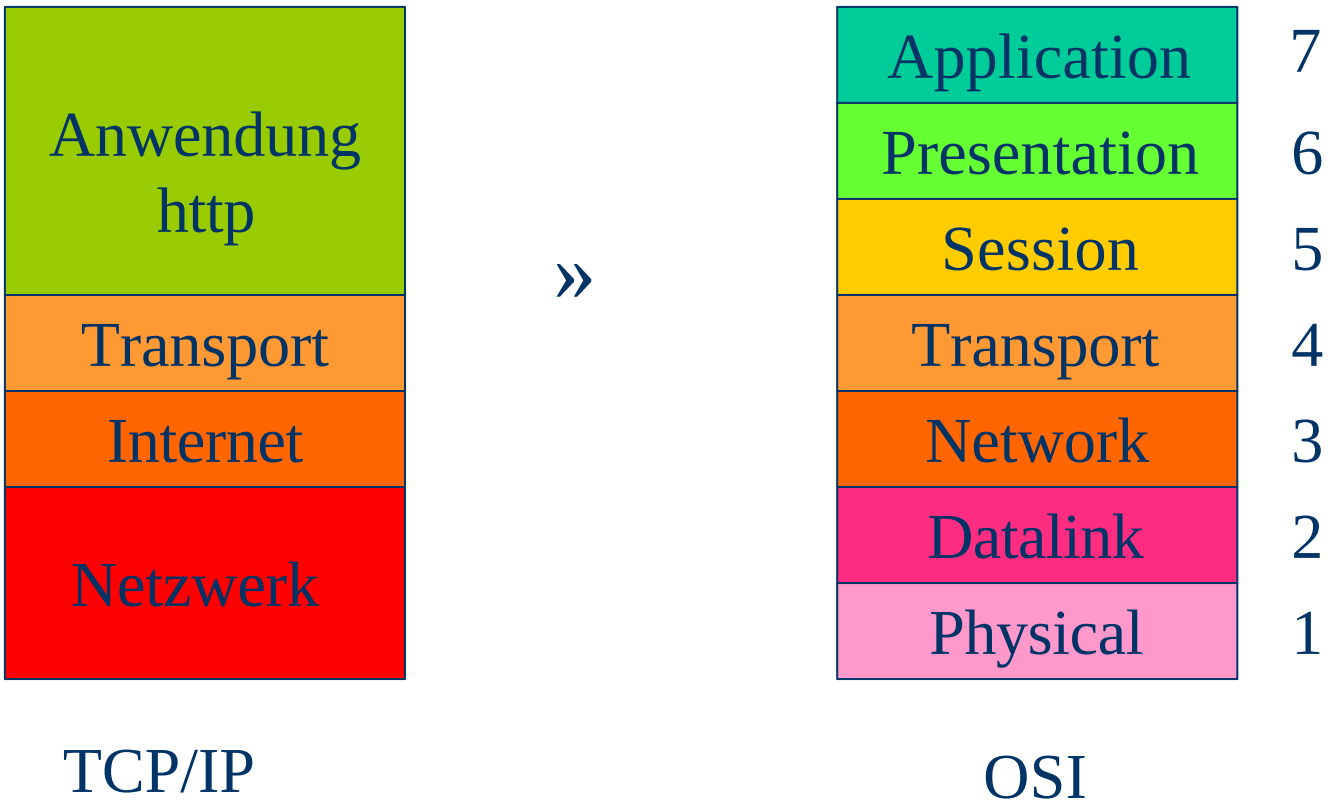
- **Transport Layer (*Transportschicht*):**
  - Abbildung zwischen Nachrichten und Packeten
  - TCP, UDP, ICMP
- **Network Layer (*Vermittlungsschicht*):**
  - Transport von Paketen zwischen beliebigen Endpunkten, Adressierung, Routing
  - IP, IPX

OSI Schichten

- **Link Layer (*Sicherungsschicht*):**
  - Realisierung eines zuverlässigen Übertragungskanals
- **Physical Layer (*Bitübertragungsschicht*):**
  - Transport von unformatierten Bit-Sequenzen
  - Ethernet, Token Ring, ATM

- "**A**ll **p**eople **s**eem **t**o **n**eed **d**ata **p**rocessing"
- "**P**lease **d**o **n**ot **t**hrow **s**alami **p**izza **a**way "

TCP/IP Referenzmodell



TCP/IP Schichten

- **Anwendungsschicht**:
  - umfasst alle Protokolle der Netzwerkinfrastruktur, für den Austausch anwendungsspezifischer Daten
- **Transportschicht**:
  - stellt eine Ende-zu-Ende Verbindung her
  - stellt Verbindungen zum gesicherten Versenden von Datenströmen zwischen jeweils zwei Netzwerkteilnehmern (TCP)

TCP/IP Schichten

- **Internetschicht**:
  - ist für die Weitervermittlung von Paketen und die Wegewahl (Routing) zuständig. Punkt-zu-Punkt-Verbindungen betrachtet
  - *Internet Protocol* (IP), das einen unzuverlässigen, verbindungslosen Paketauslieferungsdienst bereitstellt.
- **Netzwerkschicht**:
  - enthält keine Protokolle der TCP/IP-Familie.
  - Platzhalter für verschiedene Techniken zur Punkt zu Punkt Datenübertragung
  - Zusammenschließen verschiedene Subnetze

BerkeleySockets

- 4.2 BSD enthielt erstmals Sockets (1983)
- *Open-Read-Write-Close* Semantik, wie bei Dateien
- Interprozesskommunikation über Rechner- und Betriebssystemgrenzen
- Kommunikationziel ist transparent
- Berkeley Socket API ist heute Quasistandard

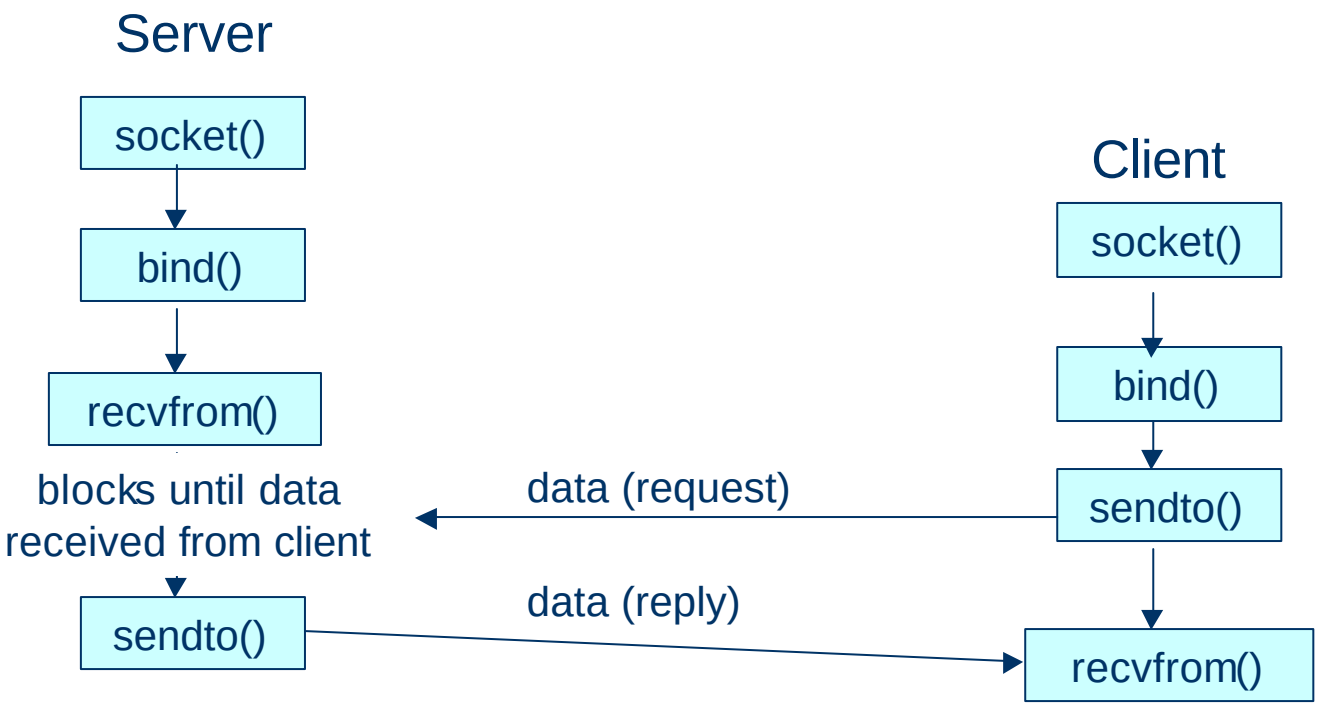
Sockets

- definiert durch 4 Werte
  - Adresse des lokalen Rechners
  - Port Nummer des lokalen Rechners
  - Adresse des entfernten Rechners
  - Port Nummer des entfernten Rechners
- Adressfamilie
- API hat synchrones Verhalten

Verbindungslos

- Verbindungslos (connectionless)
  - *User Datagram Protocol* (UDP)
  - keine Bestätigung für gesendete Nachrichten im Protokoll
  - keine Garantie für Nachrichtenreihenfolge
  - Mehrere Empfänger möglich; *Broad-/Multicast*
  - RFC 768, 1980

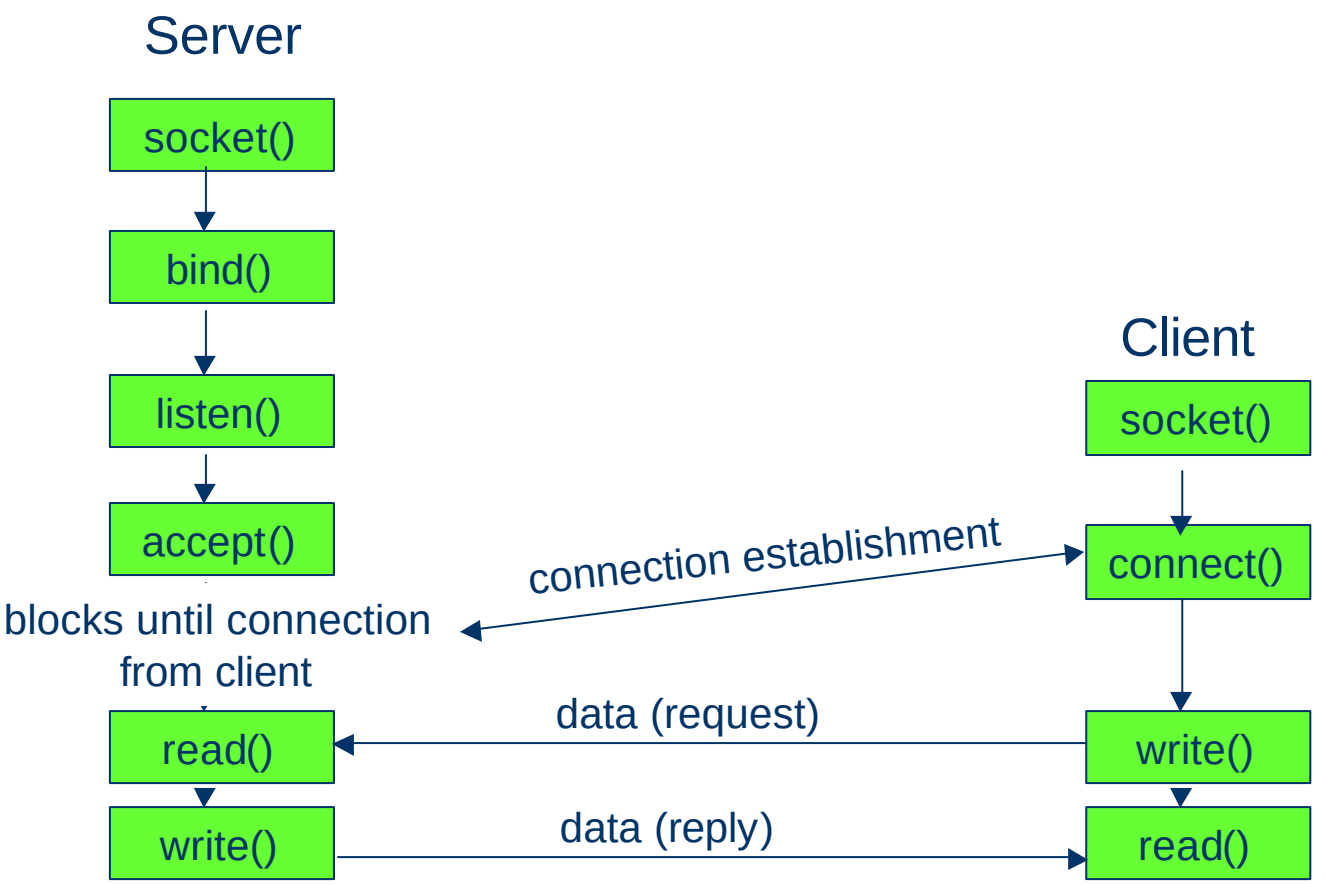
4.3 BSD UNIX-Verbindungslos



Verbindungsorientiert

- Verbindungsorientiert
  - *Transmission Control Protocol* (TCP)
  - virtuelle Verbindung zwischen **2 Sockets**
  - Nachrichten werden durch das Protokoll bei erfolgreichem Empfang bestätigt
  - Flusssteuerung
  - Timeout bei abgebrochenen Verbindungen
  - zuverlässige Kommunikation
  - RFC 793, 1981

4.3 BSD UNIX-Verbindungsorientiert



Windows vs. Unix

Windows Sockets

#include <winsock2.h>

Berkley Sockets

#include <sys/socket.h>

Link: ws2_32.lib

WSAStartup()

WSACleanup()

Windows Socket API

```
int WSAStartup(WORD wVersionRequested,
               LPWSADATA lpWSAData );
```

- Winsock Bibliothek laden: ws2_32.dll
- MAKEWORD(Major,Minor)
  - z.B. MAKEWORD(1,1) -> 1.1
  - WINSOCK_VERSION -> 2.2
  - MAKEWORD(Major,Minor) -> MSDN
- im Fehlerfall (>0) kein WSAGetLastError() bzw. GetLastError()
- `int WSACleanup(void)`

socket

- **SOCKET socket(int** *af*, **int** *type*, **int** *protocol* **);**
  - Windows
  - (unsigned) int32 = SOCKET
  - Socket-Deskriptor → TYPE

- **int** socket(**int** domain, **int** type, **int** protocol);
  - Unix
  - Benutzbar mit Dateioperationen: read(), write(), close()

af (IPV4) **domain**

AF_INET PF_INET

#define PF_INET AF_INET

type

- SOCK_STREAM → TCP
- SOCK_DGRAM → UDP

protocol

- normalerweise ist *protocol* eindeutig für *type*
- 0 für IP (RFC 1700)

Klientenverbindung

int connect**(TYPE** *s*, **const struct sockaddr*** *name*, **int** *namelen* **);**

- *s* Deskriptor eines unverbundenen Sockets (**SOCKET vs int**)
- *name* Struktur mit Verbindungsinformationen des Servers
- *namelen* Größe der übergebenen Struktur
- erzeugt eine Verbindung mit *name* in *s*

struct sockaddr

```
struct sockaddr {
    unsigned short sa_family;
    char sa_data[14];
};
```

- Template für spezielle sockaddr-Strukturen
  - sockaddr_in IPv4
  - sockaddr_in6 IPv6

```
struct sockaddr_in{
    short sin_family;
    unsigned short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};
```

- 16 Bit Port →etc/services
- 32 Bit Host-Adresse

services

Ports

- 1-1023 für das Betriebssystem reserviert
  - erfordert Superuser Rechte
- 1024-65535 (frei) verfügbar
  - etc/services reservierte Ports
  - http://www.iana.org/assignments/port-numbers

struct in_addr

```
typedef struct in_addr {
     union {
       struct {
         u_char s_b1,s_b2,s_b3,s_b4;
       } S_un_b;
       struct {
         u_short s_w1,s_w2;
       } S_un_w;
       u_long S_addr;
     } S_un;
} in_addr;
```

Adressen

- ...s_addr Konstanten
  - INADDR_ANY jede Netzwerkinterface-Adresse → bind
  - INADDR_LOOPBACK → 127.0.0.1
  - 32 Bit vorzeichenlos, long
  - Netzwerk vs Hostdarstellung von Zahlen
    - Little Endian vs Big Endian
    - 0100007f 7f000001

Lokale Adresse binden

int bind**(TYPE** *s*, **const struct sockaddr*** *name*, **int** *namelen* **);**

- *s* Deskriptor eines ungebundenen Sockets
- *nam*e Adresse mit an den der Socket gebunden werden soll
- *namelen* Größe der übergebenen Struktur

Länge der Warteschlange

int listen**(TYPE** *s*, **int** *backlog* **);**

- *s* Deskriptor eines gebundenen und nicht verbundenen Sockets
- *backlog* maximale Länge der Schlange für wartende Verbindungen
- wenn die Warteschlange voll ist, werden weitere Verbindungen abgelehnt (connection refused)

Auf Verbindungen warten

TYPE accept**(TYPE** *s***, struct sockaddr*** *addr***, int*** *addrlen* **);**

- *s* Deskriptor eines gebundenen Sockets im Listen-Modus
- *addr* Informationen über die angenommene Verbindung
- *addrlen* Größe der Struktur
- liefert einen **neuen Socket** mit der angenommenen Verbindung

Daten übertragen

- **int send(TYPE** *s*, **const char*** *buf*, **int** *len*, **int** *flags* **);**
- **int recv(TYPE** *s*, **char*** *buf*, **int** *len*, **int** *flags* **);**
- s verbundener Socket
- *buf* Pointer auf die Daten
- len Länge des Buffers
- flags Verhalten der Funktion → 0
- Rückgabe Anzahl der gesendeten/empfangenen Bytes

Socket Beispiel

- Daytime Protokoll
- RFC 867, 1983
- Varianten
  - TCP
  - Verbindungsaufbau Port 13
  - Server sendet Zeitstempel im ASCII Format
  - **eingehenden Nachrichten werden verworfen**
  - Server beendet die Verbindung

- UDP
  - Klient sendet an Port 13 eine Nachricht
  - Inhalt ist egal
  - Server sendet den Zeitstempel an den Klienten

Klient Beispiel TCP

```
struct sockaddr_in addr; /* Win32 */
SOCKET s;
char buf[255];
WSADATA wsadata;

WSAStartup(MAKEWORD(2,0),&wsadata);
s=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
addr.sin_family=AF_INET;
addr.sin_port=htons(13);//daytime
addr.sin_addr.s_addr= INADDR_LOOPBACK;

connect(s,(struct sockaddr*)&addr,sizeof(addr));
recv(s,buf,255,0);
closesocket(s);

printf("daytime %s\n", buf);
```

daytimesimple

Server Beispiel TCP

```
struct sockaddr_in addr; /* Linux */
int s, len;

s=socket(PF_INET,SOCK_STREAM,IPPROTO_TCP);

addr.sin_family=AF_INET;
addr.sin_port=htons(13);
addr.sin_addr.s_addr=INADDR_ANY;

bind(s,(struct sockaddr*)&addr,sizeof(addr);
listen(s,2);
while((c=accept(s,NULL,NULL)))
{  /* daytime contains the current timestamp */
   send(c,daytime,strlen(daytime),0);
   close(c);
}
```

UDP

int sendto(**TYPE** *s*, **const char*** *buf*, **int** *len*, **int** *flags*, **const struct** sockaddr* *to*, **int** *tolen*);

- *s* Socket (möglicherweise verbunden)
- *buf* zu sendene Daten
- *len* Länge der Daten
- *flags* Verhalten
- *to* Optionale Zieladresse (abhängig von *s*)
- *tolen* Größe von *to*

Warten auf ein UDP-Nachricht

int recvfrom**(TYPE** *s***, char*** *buf***, int** *len***, int** *flags***,**
struct sockaddr***** *from***, int*** *fromlen* **);**

- wartet auf eine Nachricht an *s*
- *s* gebundener Socket
- *buf* Empfangspuffer
- *len* Größe von *buf*
- *flags* Verhalten
- *from* optional Adresse des Senders
- *fromlen* Größe von *from*

Hilfsfunktionen

- **int** getpeername**(TYPE** *s*, **struct sockaddr*** *name*, **int*** *namelen* **);**
  - liefert die verbundene Gegenstelle des sockets
- **int** getsockname**(TYPE** *s*, **struct sockaddr*** *name*, **int*** *namelen* **);**
  - liefert lokal verbundene Adresse des Sockets

Hilfsfunktionen

- `unsigned long htonl(unsigned long);`
- `unsigned short htons(unsigned short);`
  - Host-Ordnung zu Netzwerk-Ordnung

- `unsigned long ntohl(unsigned long);`
- `unsigned short ntohs(unsigned short);`
  - Netzwerk-Ordnung zu Host-Ordnung

Hilfsfunktionen II

- **unsigned long** inet_addr(
  const char* *cp*);
- liefert eine IP-Adresse als 4 Byte Zahl
- *cp* IP Adresse im X.X.X.X Format

```
addr.sin_addr.S_un.S_addr=
        inet_addr("127.0.0.1");
```

- s_addr -> S_un.S_addr
- addr.sin_addr.s_addr=inet_addr(...);

- **char*** inet_ntoa(**struct** in_addr *in*);
  - liefert IP Adresse im X.X.X.X Format

Nichtblockierende Socket API

int select(int *nfds*, **fd_set*** *readfds*, **fd_set*** *writefds*, **fd_set*** *exceptfds*, **const struct timeval*** *timeout* **);**

- ermittelt den Status von Sockets (Deskriptoren unter UNIX), liefert Anzahl der signalisierten Sockets
- ein signalisierter Socket erlaubt Zugriff ohne Blockierung
- *readfds* Lesbar
- *writefds* Schreibar
- *exceptfds* Fehler
- *timeout* optionale Wartezeit

Makros für fd_set

- **FD_CLR(*s*, **set*)**
  - löscht *s* aus dem Set
- **FD_ISSET(*s*, **set*)**
  - wurde *s* signalisiert
- **FD_SET(*s*, **set*)**
  - fügt *s* dem Set hinzu
- **FD_ZERO(**set*)**
  - löscht das Set

getaddrinfo

```
int getaddrinfo(const char* nodename,
                const char* servname,
                const struct addrinfo* hints,
                struct addrinfo** res );
```

- Namensauflösung (DNS)
- liefert eine verkettete Liste von *addrinfo*-Strukturen
- *nodename* -> IP Adresse oder Name
- *servname* Protokolltyp (http, ftp,.. oder Port)
  - /etc/services
- *hints* gewünschte Verbindungstyen (Familie, Typ, Protokoll)

struct addrinfo

```
typedef struct addrinfo {
  int ai_flags;
  int ai_family;
  int ai_socktype;
  int ai_protocol;
  size_t ai_addrlen;
  char* ai_canonname;
  struct sockaddr* ai_addr;
  struct addrinfo* ai_next;
} addrinfo;
```

echoc.c

getaddrinfo.c