Socketprogrammierung

Socketprogrammierung in C/C++
Christian Benjamin Ries

21.02.2008

Agenda

- Ziel
- Einführung
- Netzwerkprotokolle
- Was sind Sockets?
- Funktionen für Sockets
- Extensions
- Fragen, bitte jederzeit stellen!

Einführung: Ziel

- Grobe Unterschiede zwischen TCP & UDP
- Ermittlung des Hostnamens einer IP-Adresse
- Realisierung einer einfachen Client-Server Kommunikation

Einführung

Sockets bieten die Möglichkeit protokollunabhängig bidirektionale Kommunikationsverbindungen aufzubauen:

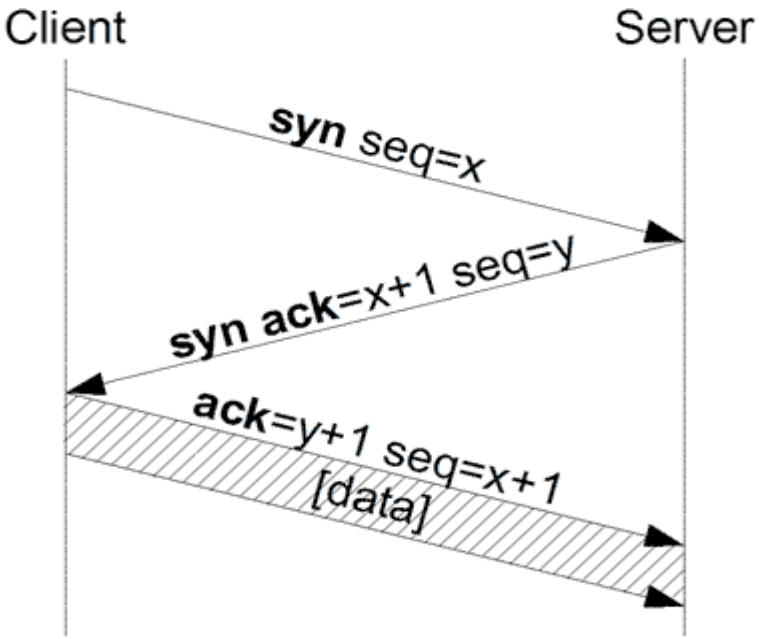
Zu unterscheiden sind...

- **verbindungsorientierte** und **verbindungslose**: Muss eine Verbindung zwischen zwei Knoten bestehen oder nicht?
- **stream-** und **paketorientiert**: Kontinuierlicher Datenstrom oder Einzelnachrichten?
- **zuverlässig** und **nicht zuverlässig**: Können Nachrichten verloren gehen, reorganisiert werden, doppelt vorkommen, etc.?

Netzwerkprotokolle

- Die wohl bekannsteten Protokolle sind TCP & UDP in Verbindung mit IP
- Anwendungsbeispiele:
  - TCP (virtuelle Verknüpfung zwischen Paketen)
    - SMTP, HTTP, TELNET => Beispiel zeigen!
  - UDP (einzelne Datagramme)
    - DNS, ROUTE (RIP), DHCP, NTP

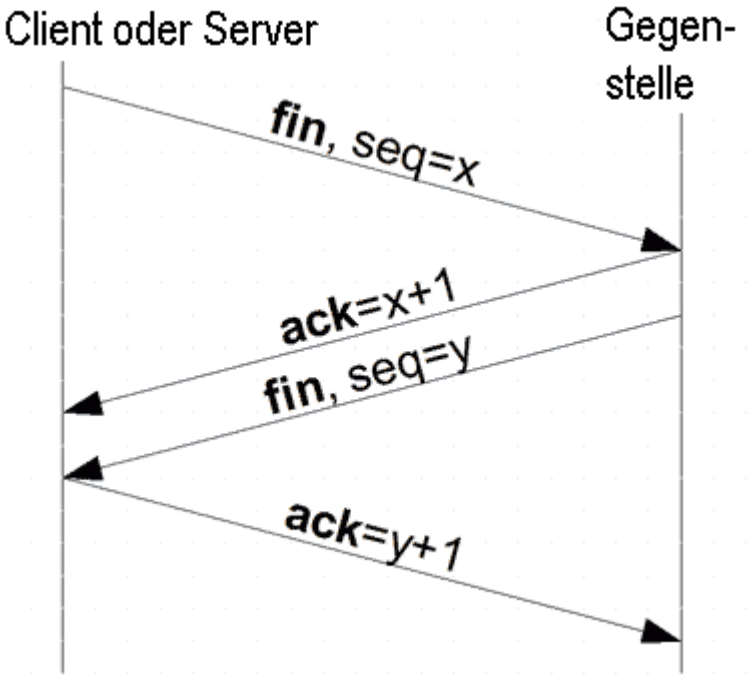
TCP Handshake



1. SYN-SENT	→	<SEQ=100><CTL=SYN>	→	SYN-RECEIVED
2. SYN/ACK-RECEIVED	←	<SEQ=300><ACK=101><CTL=SYN,ACK>	←	SYN/ACK-SENT
3. ACK-SENT	→	<SEQ=101><ACK=301><CTL=ACK>	→	ESTABLISHED

Vorteile:
- Durch die SYN/ACK-Flags kann eine verlustfreie Verbindung aufgebaut werden.
- Die Datenübertragungsgeschwindigkeit kann dem Medium angepasst werden.

TCP Teardown

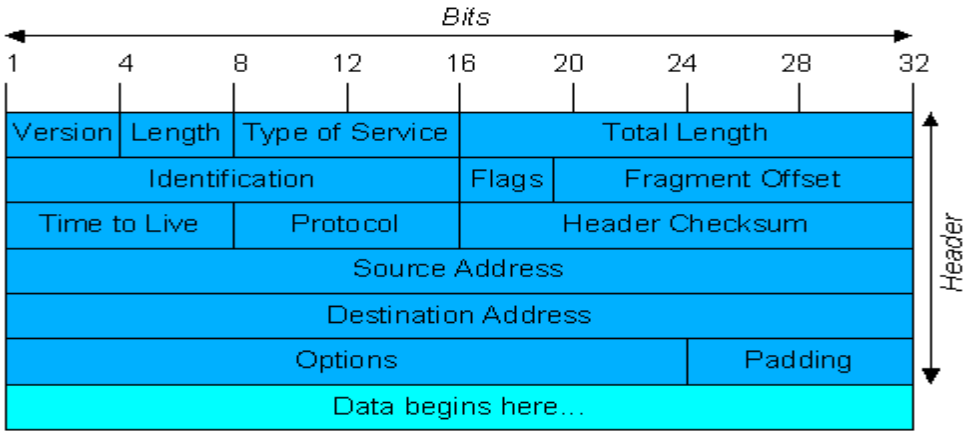


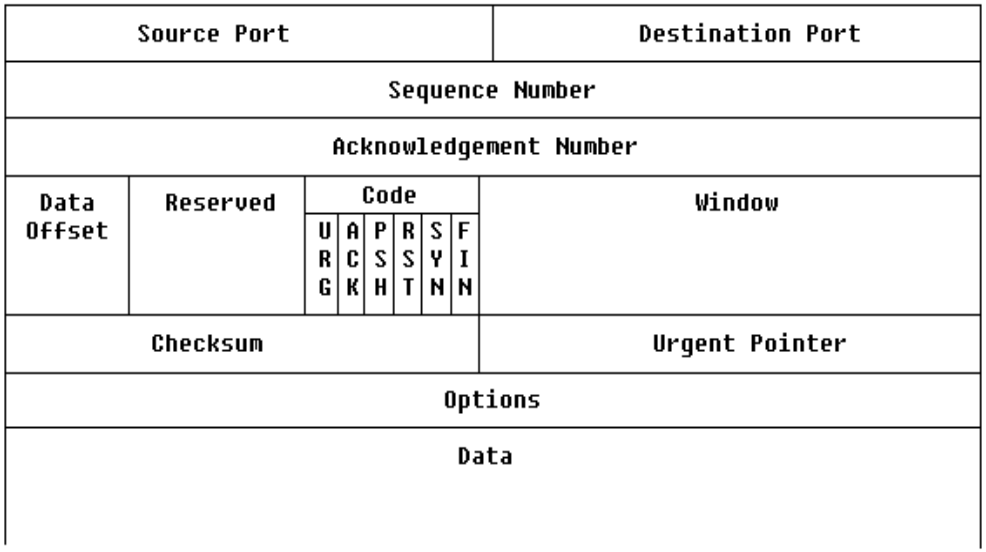
1. FIN-SENT	→	<SEQ=100><CTL=FIN>	→	FIN-RECEIVED
2. FIN/ACK-RECEIVED	←	<SEQ=300><ACK=101><CTL=FIN,ACK>	←	FIN/ACK-SENT
3. ACK-SENT	→	<SEQ=101><ACK=301><CTL=ACK>	→	CLOSED

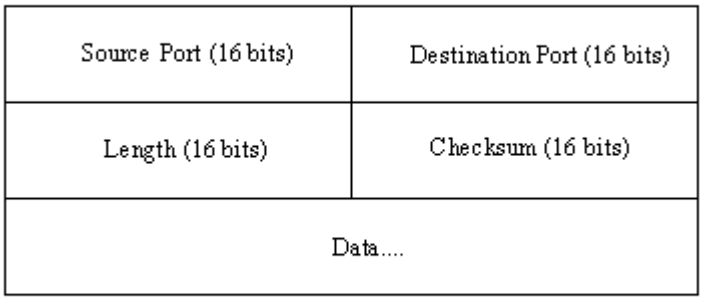
UDP

- Besitzt keinen Verbindungsaufbau/-abbau.
- Sendet die Daten auf gut Glück.
- Fehlerüberprüfungen müssen von höheren Schichten übernommen werden, dadurch mehr Aufwand bei der Anwendungsentwicklung.
- weniger Datenverkehr

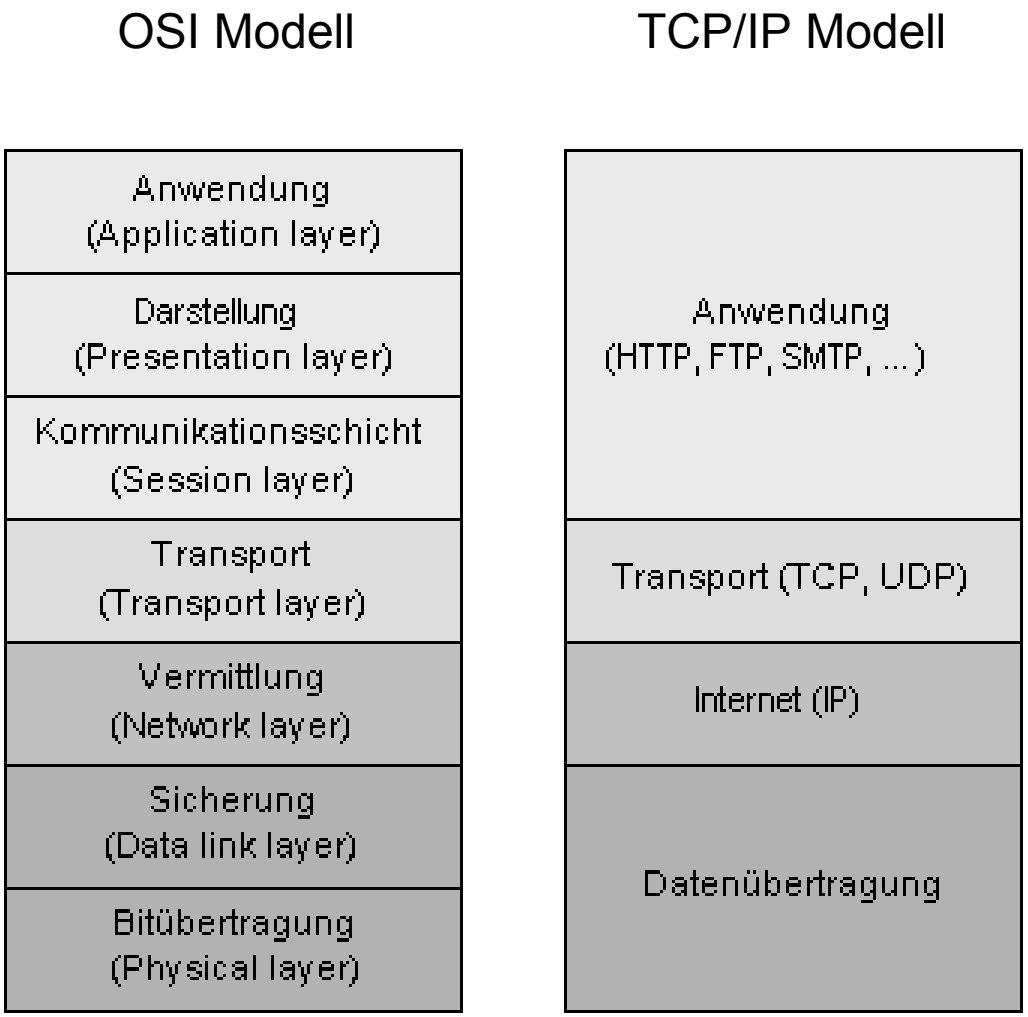
TCP / UDP Header







TCP/IP Referenzmodell



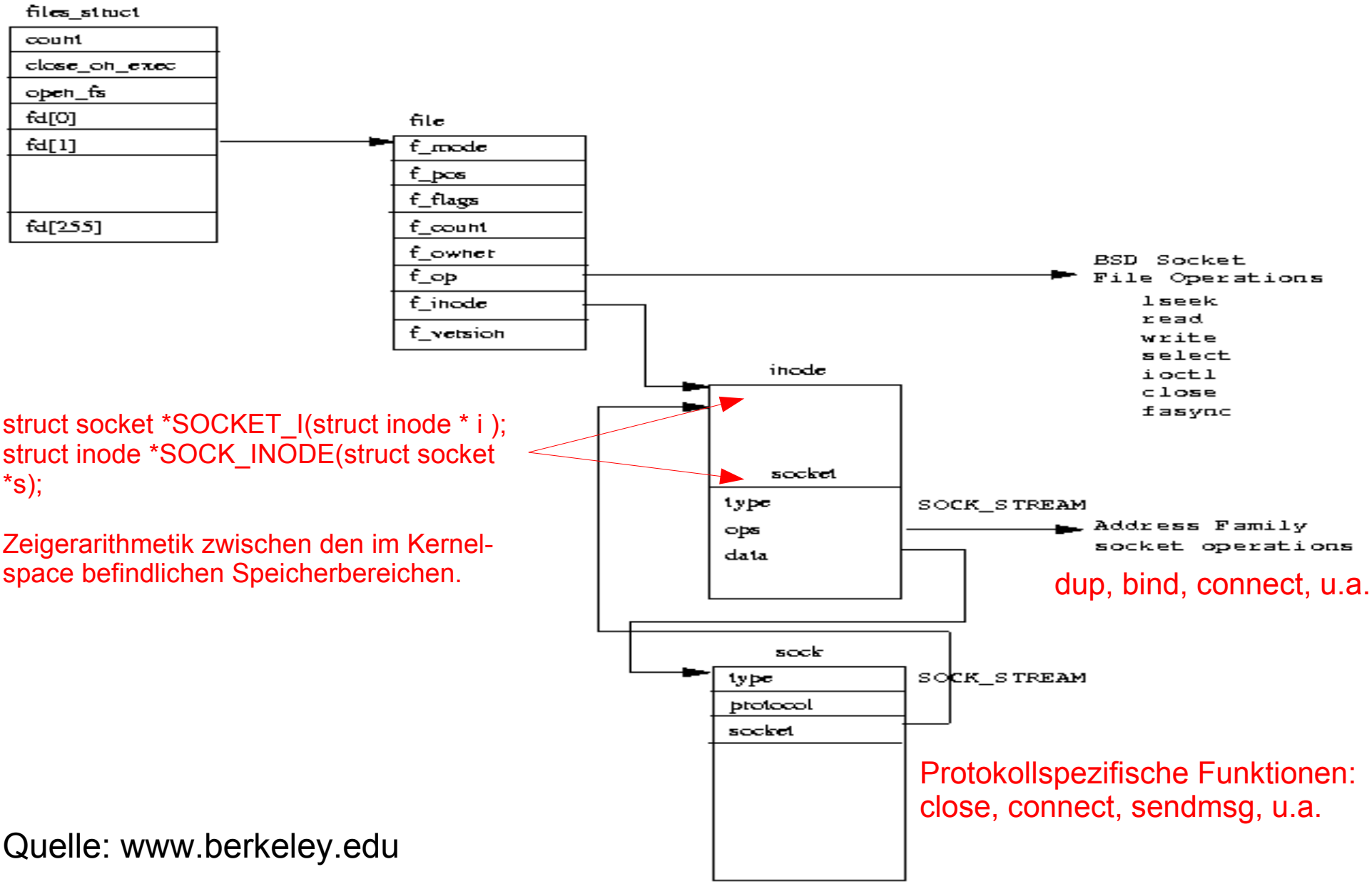
- Es besteht eine Peer-to-Peer Verbindung zwischen den Schichten (P2P).
- Wir beschäftigen uns in diesem Vortrag nur mit der Anwendungsschicht, doch füllen die darunterliegenden Schichten mit Daten.

Socket

#include <sys/socket.h>
int socket(int domain, int type, int protocol);

- eine Art Filedeskriptor im VFS (Virtual File System)
  - Vgl. PIPEs, FIFOs, System V IPC (Semaphoren, Message queues, Shared Memory)
  - Dadurch kann mit den bekannten Lese-/Schreibzugriffen gearbeitet werden.

Socket: Interna



struct socket *SOCKET_I(struct inode * i);
struct inode *SOCK_INODE(struct socket *s);

Zeigerarithmetik zwischen den im Kernel-space befindlichen Speicherbereichen.

Quelle: www.berkeley.edu

Socket: API

- **socket:** Kreiert einen Socket. (ein Telefon kaufen)
- **bind:** Bindet den Socket an eine Adresse/Port auf dem Server. (Eine Telefonnummer zuweisen)
- **listen:** Definiert die Menge an max. wartenden Clients. (Länge der Warteschleife)
- **accept:** Verbindungen akzeptieren. (Ein Telefonat annehmen)
- **connect:** Zu einem Server verbinden. (Jmd. anrufen)
- **send, sendto:** Nachrichten versenden. (Sprechen)
- **recv, recvfrom:** Nachrichten empfangen. (Zuhören)
- **shutdown:** Die Verbindung beenden. (Auflegen)

Socket: TCP Server

Folgenden Aufrufe benötigt ein Server:

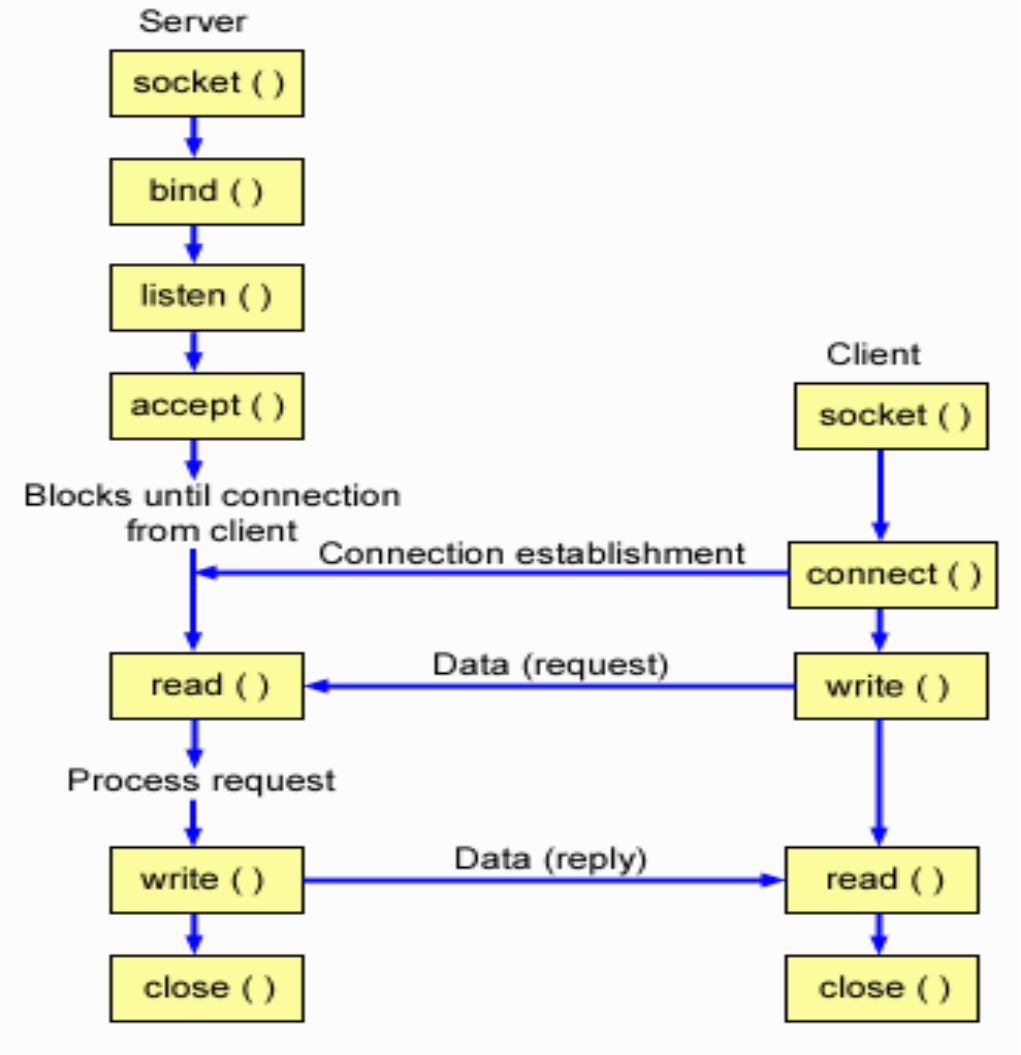
- **socket:** Socket erstellen.
- **bind:** Bindet die Adresse des Servers an den Socket.
- **listen:** Max. Anzahl an möglich wartenden Clients.
- **accept:** Verbindungen akzeptieren.
- **send, recv:** Nachrichten empfangen/versenden.
- **shutdown:** Verbindung beenden.
- **close:** Zugewiesenden Daten im Kernel freigeben.

Socket: TCP Client

Folgenden Aufrufe benötigt ein Client:

- **socket:** Socket erstellen.
- **connect:** Eine Verbindung herstellen.
- **send, recv:** Nachrichten empfangen/versenden.
- **shutdown:** Verbindung beenden.
- **close:** Zugewiesenden Daten im Kernel freigeben.

Socket: TCP



Quelle: IBM

int socket(...)

#include <sys/socket.h>
int socket(int domain, int type, int protocol);

- **int domain**
  - AF_INET: Internetkommunikation
  - AF_UNIX: Lokale Kommunikation über Dateien
- **int type**:
  - SOCK_STREAM: TCP Verbindung
  - SOCK_DGRAM: UDP Verbindung
- **int protocol**: (meist 0 = IP)
  - siehe /etc/protocols

Beispiel:
socket(AF_INET, SOCK_RAW, icmp);

Bau eines eigenes PING Programms.

int bind(...)

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

- **int sockfd**: Socket id von socket(...).
- **struct sockaddr * my_addr**:
  - Protokollspezifische Adressstruktur
  - Im Internetprotokoll besteht die Adresse aus: IPADDR & PORT
- **int addrlen**: Größe in Bytes die my_addr belegt.
- $ man 3 bind

Wie ist die Struktur '**sockaddr**' definiert ?

struct sockaddr_in

```
struct sockaddr_in        (sockaddr_in6 bei Ipv6)
   {
     short int             sin_family;     /* Address family             */
     unsigned short int    sin_port;       /* Port number                */
     struct in_addr        sin_addr;       /* Internet address           */
     unsigned char         sin_zero[8];    /* Same size as struct sockaddr */
   };
```

Definiert in '/usr/include/sys/socket.h'.

sin_port und sin_addr müssen in Network-Byte-Order sein!

Network-Byte-Order

- zwei verschiedene Ordnungen gibt es:
  - Little-Endian: Kleines Ende zuerst!
  - Big-Endian: Großes Ende zuerst!
- Die Network-Byte-Order ist nach Big-Endian orientiert!

- siehe '**byteorder**' Beispiel!

Network-Byte-Order

```
#include <arpa/inet.h>
   uint32_t htonl( uint32_t hostlong );
   uint16_t htons( uint16_t hostshort );
   uint32_t ntohl( uint32_t netlong );
   uint16_t ntohs( uint16_t netshort );
```

htonl: Host to Network Long
htons: Host to Network Short
ntohl: Network to Host Long
ntohs: Network to Host Short

16 bzw. 32 Bits = für Ports und 32 Bit IP-Adressen

int listen(...)

```
#include <sys/socket.h>
int listen(int s, int backlog);
```

- **int s**: Socket id von socket(...).
- **int backlog**: Anzahl der Clients die sich gleichzeitig verbinden dürfen.
- $ man 3 listen

int accept(...)

```
#include <sys/socket.h>
int accept(int socket, struct sockaddr * address,
                       socklen_t * address_len);
```

- **int socket**: Socket id von socket(...).
- **struct sockaddr *address**: Zeiger auf die Adressstruktur.
- **socklen_t * address_len**: Größe der Struktur
- $ man 3 accept

int connect(...)

```
#include <sys/socket.h>
int connect( int socket, const struct sockaddr *address,
                                    socklen_t address_len );
```

- **int socket**: Socket id von socket(...).
- **struct sockaddr *address**: Zeiger auf die Adressstruktur.
- **socklen_t address_len**: Größe der Struktur
- $ man 3 connect

int send/sendto(...)

```
#include <sys/socket.h>
ssize_t send( int socket, const void * buffer, size_t length, int flags );
ssize_t sendto( int socket, const void *buffer, size_t length, int flags,
                const struct sockaddr *dest_addr, socklen_t dest_len );
```

- **int socket**: Socket id von socket(...).
- **void *buffer:** Die Daten die versendet werden sollen, kann alles drin stehen.
- **size_t length:** Größe der Daten in Bytes.
- **int flags:** MSG_MORE, MSG_DONTROUTE
- **...**: Die Parameter bei sendto(...) sind dieselben wie bei bind(...), diese Funktion findet meist bei UDP Applikationen Anwendung.
- $ man 3 send $ man 3 sendto

int recv/recvfrom(...)

```
#include <sys/socket.h>
ssize_t recv( int socket, void * buffer, size_t length, int flags );
ssize_t recvfrom( int socket, void *buffer, size_t length, int flags,
                  struct sockaddr *addr, socklen_t len );
```

- int **socket**: Socket id von socket(...).
- **void *buffer:** Die Daten die versendet werden sollen, kann alles drin stehen.
- **size_t length:** Größe der Daten in Bytes.
- **int flags:** MSG_WAITALL, MSG_PEEK (mark unread, read again)
- **...**: Die Parameter bei recvfrom(...) sind dieselben wie bei bind(...), diese Funktion findet meist bei UDP Applikationen Anwendung.
- $ man 3 recv $ man 3 recvfrom

int shutdown(...)

```
#include <sys/socket.h>
int shutdown( int socket, int how );
```

- **int socket**: Socket id von socket(...).
- **int how:** Gibt an wie die Verbindung beendet werden soll.
  - SHUT_RD: Beendet die Möglichkeit vom Socket zu lesen.
  - SHUT_WR: Beendet die Möglichkeit auf den Socket zu schreiben.
  - SHUT_RDWR: Beides
- $ man 3 shutdown

Extensions

Arten von Client-Server Verbindungen:

- Einzelverbindungen, einzelne Direktverbindung
  - langweilig!!!
- Anzahl von X Verbindungen
  - schon spannender!!!

Extensions

Wie verwaltet man mehrere Verbindungen?

- Im Grunde gibt es zwei Verfahren für diese Aufgabe:
  - Polling (auch langweilig!)
  - Blockierend (nehmen wir das!)

int select(...)

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *timeout);
      void FD_CLR(int fd, fd_set *fdset);      int FD_ISSET(int fd, fd_set *fdset);
      void FD_SET(int fd, fd_set *fdset);      void FD_ZERO(fd_set *fdset);
```

- **int nfds**: höchste Socket ID + 1
- **fd_set * readfds, writefds, errorfds:** Bitfeld mit maskiertem Bit für die jeweilige aktuell geänderte Socket ID.
- **struct timeval *timeout:**
  - struct timeval { long int tv_sec; long int tv_usec; };
  - Sekunden und Mikrosekunden bis select(...) von selbst zurückkehrt.
- mhmmm ein Beispiel wäre wohl an dieser Stelle sinnig???

Hostname/IP ermitteln

```
#include <netdb.h>
    struct hostent *gethostbyname( const char *name );
    struct hostent *gethostbyaddr( const char *addr, int len, int type = AF_INET );

struct hostent {
        char    *h_name;        /* Offizieller Name des Rechners */
        char    **h_aliases;    /* Aliasliste */
        int     h_addrtype;     /* Host-Adress-Typ */
        int     h_length;       /* Adresslänge */
        char    **h_addr_list;  /* Adressliste */
    }
    #define h_addr  h_addr_list[0]  /* für Abwärtskompatibilität */
```

Siehe Beispiel 'gethostbyaddr'!

Optionen

```
#include <sys/socket.h>
    int setsockopt( int socket, int level, int option_name,
                        const void * option_value, socklen_t option_len );
```

- **int level:** auf welcher Ebende die Option gesetzt werden soll
- **int option_name:** SO_REUSEADDR, SO_DEBUG
- **void * option_value:** Wert, auf die die Option gesetzt werden soll.
- **socklen_t option_len:** Größe des 'option_value' in Byte.

```
cont int y = 1;
setsockopt( socketid, SOL_SOCKET, SO_REUSEADDR, &y, sizeof(int));
```

$ man 3 setsockopt

Ende

Fragen ???