1 Netzwerkkommunikation und Byteorder

- Wiederholung: Byteorder

	0	1	2	3
big endian	11	aa	bb	cc
little endian	cc	bb	aa	11

0x11aabbcc

- Kommunikation zwischen Rechnern verschiedener Architekturen (z.B. Intel Pentium (litte endian) und Sun Sparc (big endian))
- **`htons`**, **`htonl`**: Wandle Host-spezifische Byteordnung in Netzwerk-Byteordnung (big endian) um
(**`htons`** für **`short int`**, **`htonl`** für **`long int`**)
- **`ntohs`**, **`ntohl`**: Umgekehrt

2 Wiederholung: Streams

- Zwei Möglichkeiten für IO
  - Benutzung der Systemschnittstelle durchFiledeskriptoren (**`open,read,write,close`**)
  - Benutzung der C-Bibliothek durch Streams
- Streams: I/O durch **`FILE`**-Objekte, z.B. stdin, stdout, stderr
- Unterschiede
  - Stream-I/O ist gepuffert, Leeren des Puffers mit **`fflush(fp)`**
  - Stream-I/O ist portabel (Teil des ANSI-C-Standards)
- Wichtige Funktionen
  - fprintf, fscanf, fflush, feof, fopen, fclose, fgetc, fputc, fgets, fileno

2 Streams Beispiel

```
#include <stdio.h>
#include <stdlib.h> /* Definition von EXIT_FAILURE */

FILE *fp;

fp = fopen("/tmp/hallo", "w");
if (fp == NULL) {
    fprintf(stderr, "Error opening file.\n");
    exit(EXIT_FAILURE);
}

fprintf(fp, "Hello World!\n");
fclose(fp);
```


3 Socket - Erzeugung

- `s = socket(domain, type, protocol)`
- Domain, z.B.
  - AF_INET: Internet
  - AF_UNIX: Unix Filesystem
  - AF_APPLETALK: Appletalk Netzwerk
- Type in AF_INET und AF_UNIX Domain:
  - SOCK_STREAM: Stream-Socket (bei AF_INET TCP)
  - SOCK_DGRAM: Datagramm-Socket (bei AF_INET UDP)
  - SOCK_RAW
- Protokoll
  - Default-Protokoll für Domain/Type Kombination: 0
    (z.B. INET/STREAM -> TCP) (siehe `getprotobyname(3)`)

4 Binden von Sockets

- Was wird gebunden?
  - locale und remote IP-Adressen, lokale und remote Ports
  - Portnummern sind eindeutig für einen Rechner und ein Protokoll
  - Portnummern < 1024: privilegierte Ports für root (z.B. www=80, Mail=25, finger=79)
  - Portnummern sind 16 Bit, d.h. kleiner als 65535
- Eine Verbindung ist eindeutig gekennzeichnet durch
  - lokale Adresse,Port und remote Adresse,Port
- **bind** bindet an lokale IP-Adresse + Port
  - bind(s, name, namelen)
  - name: Protokollspezifische Adresse
  - namelen: Größe der Adresse in Byte

4 Lokales Binden eines TCP Socket

- **INADDR_ANY**: wenn Socket auf allen lokalen Adressen (z.B. allen Netzwerkinterfaces) Verbindungen akzeptieren soll
- **sin_port = 0**: wenn die Portnummer vom System ausgewählt werden soll
- Adresse und Port muß in Netzwerk-Byteorder vorliegen
- Beispiel

```
#include <sys/types.h>
#include <netinet/in.h>
...
struct sockaddr_in sin;
...
s = socket(AF_INET, SOCK_STREAM, 0);
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = htonl(INADDR_ANY);
sin.sin_port = htons(MYPORT);
bind(s, (struct sockaddr *) &sin, sizeof sin);
```

5 Socket Adressen

- Socket-Interface (`<sys/socket.h>`) ist protokoll-unabhängig

```
struct sockaddr {
    sa_family_t     sa_family;        /* Adressfamilie */
    char            sa_data[14];      /* Adresse */
};
```

- Internet-Protokoll-Familie (`<netinet/in.h>`) verwendet

```
struct sockaddr_in {
    sa_family_t         sin_family;    /* = AF_INET */
    in_port_t           sin_port;      /* Port */
    struct in_addr      sin_addr;      /* Internet-Adresse */
    char                sin_zero[8];   /* Füllbytes */
}
```


6 Binden an remote Adresse (Server)

- Server:
  - `listen` stellt ein, wieviele Verbindungen gepuffert sein können (d.h. auf ein `accept` wartend)
  - `accept` akteptiert Verbindung, d.h. erzeugt neuen Socket und bindet diesen an remote Adresse + Port

```
struct sockaddr_in from;
  ...
listen(s, 5);                /* Allow queue of 5 connections */
fromlen = sizeof(from);
newsock = accept(s, (struct sockaddr *) &from, &fromlen);
```

7 Binden an remote Adresse (Client)

- Client bindet mit connect an die remote Adresse, dadurch wird auch eine lokale Bindung hergestellt

```
struct sockaddr_in server;
  ...
connect(s, (struct sockaddr *)&server, sizeof server);
```


8 Lesen und Schreiben auf Sockets

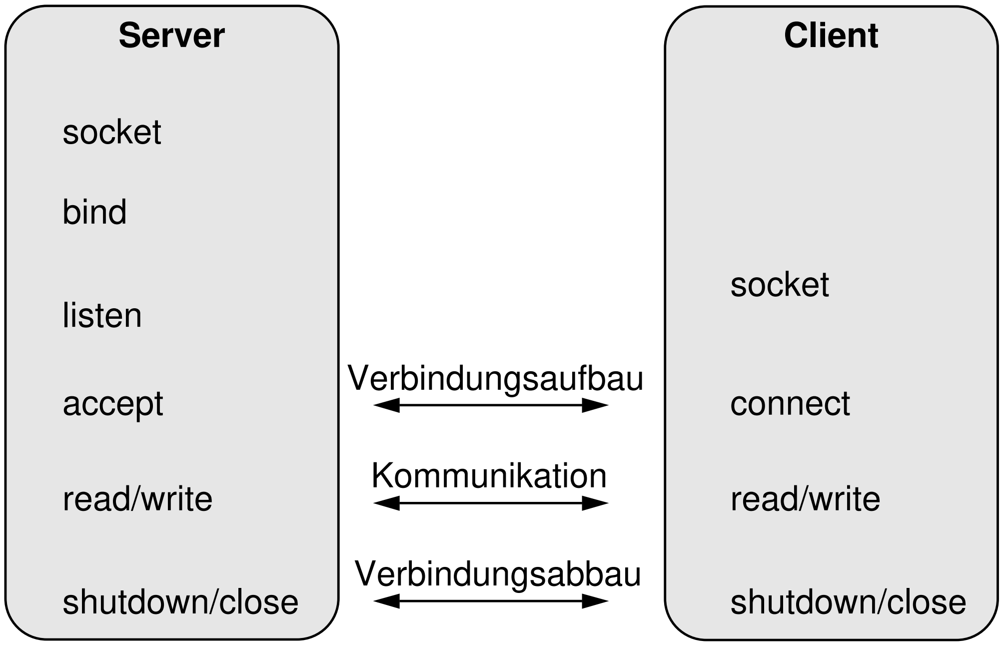
- mit **read**, **write**
- Beispiel: Server, der alle Eingaben wieder zurückschickt

```
fd = socket(AF_INET, SOCK_STREAM, 0); /* Fehlerabfrage */

name.sin_family = AF_INET;
name.sin_port = htons(port);
name.sin_addr.s_addr = htonl(INADDR_ANY);

bind(fd, (const struct sockaddr *)&name, sizeof(name)); /* Fehlerabfrage */

listen(fd, 5); /* Fehlerabfrage */

in_fd = accept(fd, 0, 0); /* Fehlerabfrage */

for(;;) {

   n = read(in_fd, buf, sizeof(buf)); /* Fehlerabfrage */

   write(in_fd, buf, n); /* Fehlerabfrage */

}

close(in_fd);
```

9 Schließen einer Socketverbindung

- shutdown(s, how)
- how:
  - 0: verbiete Empfang
  - 1: verbiete Senden
  - 2: verbiete Senden und Empfangen

10 TCP-Sockets: Zusammenfassung



11 Duplizieren von Filedeskriptoren

- **`newfd = dup(fd)`**: Dupliziert Filedeskriptor fd, d.h. Lesen/Schreiben auf newfd ist wie Lesen/Schreiben auf fd
- **`dup2(fd, newfd)`**: Dupliziert FD in anderen FD (newfd), falls newfd schon geöffnet ist, wird newfd erst geschlossen
- Verwenden von dup2, um stdout umzuleiten:

```
fd = open("/tmp/myoutput", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
dup2(fd,fileno(stdout));
printf("Hallo\n"); /* wird in /tmp/myoutput geschrieben */
```


12 Telnet - Tip

- Beim vt100-Terminal werden Zeilen mit <LF> ("\n") abgeschlossen
- Das telnet-Programm implementiert ein einfaches Terminal, dessen Zeilen mit <CR><LF> (= "\r\n") abgeschlossen werden

13 Netzwerk-Programmierung - Wissenswertes

- Informationen über Socket-Bindung
- Hostnamen und -adressen ermitteln

13.1 getsockname, getpeername

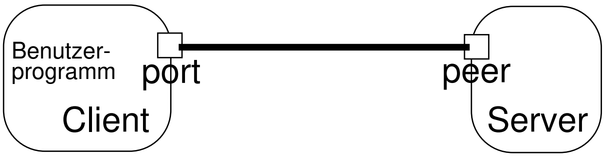
```
#include <sys/socket.h>
int getsockname(int s, void *addr, int *addrlen);
int getpeername(int s, void *addr, int *addrlen);
```

- Informationen über die lokale Adresse des Socket

```
struct sockaddr_in server;
size_t len;

len = sizeof(server);
getsockname(sock, (struct sockaddr *) &server, &len);
printf("Socket port #%d\n", ntohs(server.sin_port));
```

- Informationen über die remote Adresse des Socket

```
struct sockaddr_in server;
size_t len;

len = sizeof(server);
getpeername(sock, (struct sockaddr *) &server, &len);
printf("Socket port #%d\n", ntohs(server.sin_port));
```



13.2 Hostnamen und Adressen

- **`gethostbyname`** liefert Informationen über einen Host

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);
struct hostent {
   char     *h_name; /* offizieller Rechnername */
   char     **h_aliases; /* alternative Namen */
   int      h_addrtype; /* = AF_INET */
   int      h_length; /* Länge einer Adresse */
   char     **h_addr_list; /* Liste von Netzwerk-Adressen,
                              Abgeschlossen durch NULL */
};

#define h_addr h_addr_list[0]
```

- **`gethostbyaddr`** sucht Host-Informationen für bestimmte Adresse

```
struct hostent *gethostbyaddr(const void *addr, size_t len, int type);
```


13.3 Socket-Adresse aus Hostnamen erzeugen

```
char *hostname = "faui07a";
struct hostent *host;
struct sockaddr_in saddr;

host = gethostbyname(hostname);
if(!host) {
    perror("gethostbyname()");
    exit(EXIT_FAILURE);
}
memset(&saddr, 0, sizeof(saddr)); /* Struktur initialisieren */
memcpy((char *) &saddr.sin_addr, (char *) host->h_addr, host->h_length);
saddr.sin_family = AF_INET;
saddr.sin_port   = htons(port);

/* saddr verwenden ... z.B. bind oder connect */
```