

7. Tafelübung

- Lösung der jsh-Aufgabe
- Erläuterung der rshd-Aufgabe
- Sockets

SP I

Übung zu Systemprogrammierung I

© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000

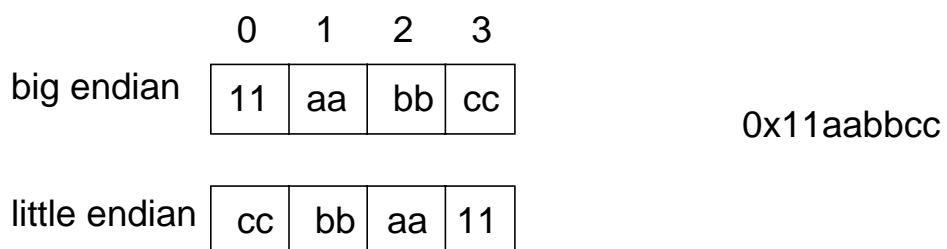
1999-12-20 14.51 / Tafelübung 7

83

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Netzwerkkommunikation und Byteorder

- Wiederholung: Byteorder



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

SP I

Übung zu Systemprogrammierung I

© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000

1999-12-20 14.51 / Tafelübung 7

84

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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 `getprotobynumber(3)`)

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

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);
```

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 */
};
```

Binden an remote Adresse (Server)

■ Server:

- ◆ `listen` stellt ein, wieviele Verbindungen gepuffert sein können (d.h. auf ein `accept` wartend)
- ◆ `accept` akzeptiert 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);
```

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);
```

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 */

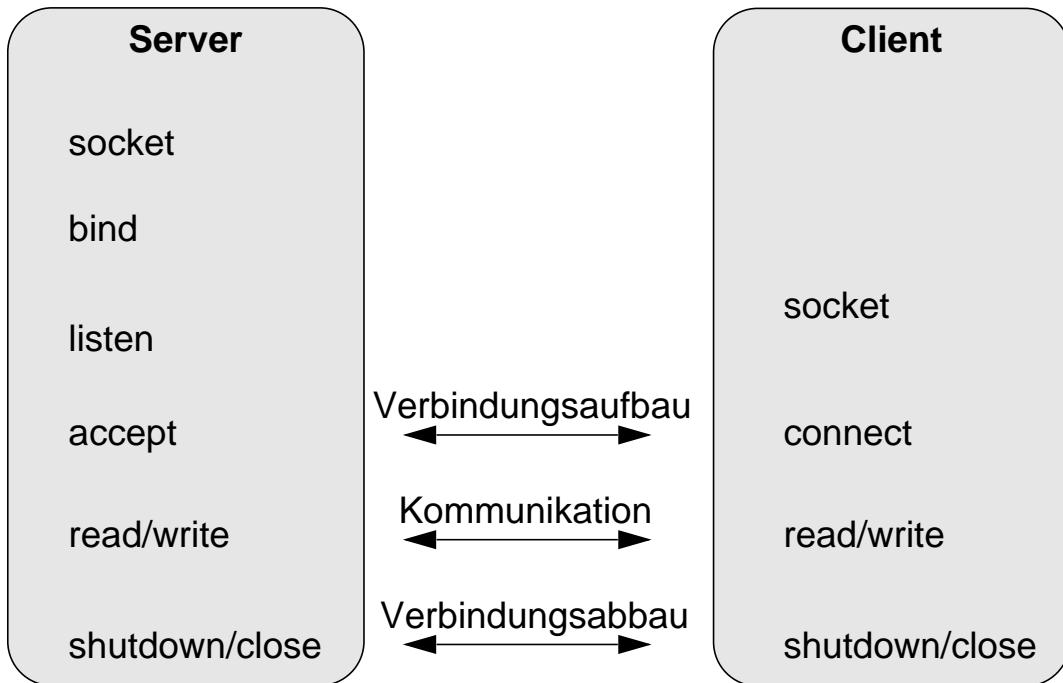
}

close(in_fd);
```

Schließen einer Socketverbindung

- `shutdown(s, how)`
- `how:`
 - ◆ SHUT_RD: verbiete Empfang
 - ◆ SHUT_WR: verbiete Senden
 - ◆ SHUT_RDWR: verbiete Senden und Empfangen

TCP-Sockets: Zusammenfassung



SP I

Übung zu Systemprogrammierung I
© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000

1999-12-20 14.51 / Tafelübung 7

93

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Sockets und UNIX-Standards

- Sockets sind nicht Bestandteil des POSIX.1-Standards
- Sockets stammen aus dem BSD-System, sind inzwischen Bestandteil von
 - ◆ BSD (-D_BSD_SOURCE)
 - ◆ SystemV R4 (-DSVID_SOURCE)
 - ◆ UNIX 95 (-D_XOPEN_SOURCE -D_XOPEN_SOURCE_EXTENDED=1)
 - ◆ UNIX 98 (-D_XOPEN_SOURCE=500)

SP I

Übung zu Systemprogrammierung I
© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000

1999-12-20 14.51 / Tafelübung 7

94

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Die Open Group (X/Open)

- Eigentümer des Markenzeichens "UNIX"
- Erstellen Spezifikationen (Systemaufruf-Schnittstellen, Programme, ...)
- Hersteller können für ihr Betriebssystem ein "Branding" erwerben
- The Single UNIX® Specification (UNIX 95)
 - ◆ enthält STREAMS, Sockets, XTI, POSIX.1, BSD und SVID Schnittstellen
 - ◆ Solaris 2.5 and 2.5.1, HP-UX 10.10, IBM AIX 4.2, Digital Unix 4
- The Single UNIX® Specification, Version 2 (UNIX 98)
 - ◆ <http://www.opengroup.org/onlinepubs/007908799/>

SP I

Übung zu Systemprogrammierung I

© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000

1999-12-20 14.51 / Tafelübung 7

95

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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 */
```

SP I

Übung zu Systemprogrammierung I

© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000

1999-12-20 14.51 / Tafelübung 7

96

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

SP I

Übung zu Systemprogrammierung I
© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000

1999-12-20 14.51 / Tafelübung 7

97

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Netzwerk-Programmierung - Wissenswertes

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

SP I

Übung zu Systemprogrammierung I
© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000

1999-12-20 14.51 / Tafelübung 7

98

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

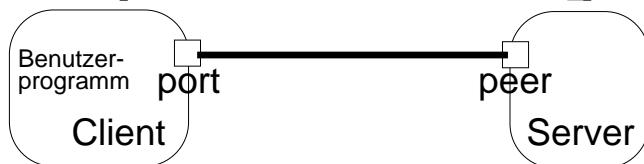
```
struct sockaddr_in server;
size_t len;

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));
```



Hostnamen und Adressen (BSD)

- `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);
```

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 */
```