

## U9 Interprozesskommunikation mit Sockets

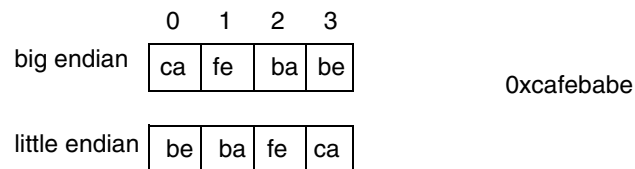
- Evaluation
- Byteorder bei Netzwerkkommunikation
- Sockets
- Verschiedenes

## U9-1 Hinweise zur Evaluation

- in Kommentaren, die sich auf einen bestimmten Übungsleiter beziehen, bitte dessen Namen **jeder Zeile** voranstellen
- Frage "eigener Aufwand zur Vor- und Nachbereitung"
  - ◆ bitte nach Vorlesung und Übung auftrennen
  - ◆ Vorlesung: den jeweiligen Wochenaufwand durch 2 teilen (2x 90 Minuten Vorlesung pro Woche, Aufwandsangabe je 90 Minuten)
  - ◆ Übung: den jeweiligen Wochenaufwand durch 4 teilen (2x2 Stunden Übung (Tafel+Rechner) pro Woche, Angabe je 45 Minuten)
- Vorlesungsevaluation: "Dozent hat Vorlesung zu ... selbst gehalten"
  - ◆ Dozenten sind Prof. Schröder-Preikschat und Dr. Jürgen Kleinöder
  - ◆ technisch bedingt wird in der Evaluation nur Prof. Schröder-Preikschat als Dozent genannt
  - ◆ bitte beide Dozenten bei der Beantwortung der Frage berücksichtigen

## U9-2 Netzwerkkommunikation und Byteorder

- Wiederholung: Byteorder



- Kommunikation zwischen Rechnern verschiedener Architekturen z. B. Intel Pentium (little 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

## U9-3 Sockets

- Endpunkte einer Kommunikationsverbindung
- Arbeitsweise: FIFO, bidirektional
- Attribute:
  - **Name** (Zuweisung eines Namens durch *Binding*)
  - **Communication Domain**
  - **Typ**
  - **Protokoll**

## 1 Communication Domain und Protokoll

- **Communication Domain** legt die **Protokoll-Familie**, in der die Kommunikation stattfindet, fest
- Protokoll-Familie legt gleichzeitig auch die Adressierungsstruktur (**Adress-Familie**) fest (unabhängig geplant, aber nie getrennt)
- das **Protokoll**-Attribut wählt das Protokoll innerhalb der Familie aus
- Communication Domains (Protokoll-Familie / Adress-Familie)
  - UNIX-Domain (PF\_UNIX / AF\_UNIX)
  - Internet-Domain IPv4 (PF\_INET / AF\_INET)
  - Internet-Domain IPv6 (PF\_INET6 / AF\_INET6)
- Werte von Protokollfamilie (PF\_) und Adress-Familie (AF\_) identisch

## 2 Internet-Domain: Socket-Namen

- über den Socket-Namen ist ein Socket innerhalb der Kommunikationsdomäne eindeutig erreichbar
- Internet-Domain: IP-Adresse + Port-Nummer
- IP-Adresse identifiziert einen Rechner
- Port-Nummer einen Prozess auf einem Rechner
  - ◆ ggf. sogar einen von mehreren von diesem Prozess angebotenen Diensten

## 2 Internet-Domain: Protokolle

- Internet Protocol - IP (Vermittlungsschicht)
  - Netzwerkprotokoll zur Bildung eines virtuellen Netzwerkes auf der Basis mehrerer physischer Netze
  - definiert Format der Datengrundeinheit - IP-Datagramm
  - unzuverlässige Datenübertragung
  - Routing-Konzepte (IP-Pakete über mehrere Zwischenstationen leiten)
  - IP-Adressen: 4 Byte bei IPv4 bzw. 16 Byte bei IPv6
- User Datagram Protocol - UDP (Transportschicht)
  - IP adressiert Rechner, UDP einen Dienst (siehe Port-Nummern)
  - Übertragung von Paketen (**sendto**, **recvfrom**), unzuverlässig (Fehler werden erkannt, nicht aber Datenverluste)
- Transmission Control Protocol - TCP (Transportschicht)
  - zuverlässige Verbindung (Datenstrom) zu einem Dienst (Port)

## 2 Internet-Domain: Adressen

- IPv4: 4 Byte
  - ◆ Notation: 4 mit '.' getrennte Byte-Werte in Dezimaldarstellung
  - ◆ z. B. 131.188.34.45
- IPv6: 16 Byte
  - ◆ Notation: acht mit ':' getrennte 2-Byte-Werte in Hexadezimaldarstellung
  - ◆ z.B.: 2001:638:a00:1e:2e0:81ff:fe58:67ab
  - ◆ in der Adresse kann einmalig '::' als Kurzschreibweise einer Nullfolge verwendet werden
  - ◆ Beispiel: IPv6 localhost-Adresse: 0:0:0:0:0:0:0:1 = ::1
- Einführung von IPv6 schleppend
  - ◆ 1998 verabschiedet, Verbreitung immer noch sehr gering
  - ◆ Software sollte sowohl IPv4 als auch IPv6 unterstützen

## 2 Internet-Domain: Port-Nummern

- bestimmten Prozess (Dienst) ansprechen (vgl. Vorl. 7-95, Tor)
- die intuitive Lösung, als Ziel eine Prozess-ID zu nehmen, hat Nachteile
  - ◆ Prozesse werden dynamisch erzeugt und vernichtet- die *PID* ändert sich dadurch
  - ◆ Ziele sollten aufgrund ihrer Funktion (Dienst) ansprechbar sein
  - ◆ Prozesse können mehrere Dienste anbieten (vgl. *inetd*)
- Lösung: **Port** als "abstrakte Adresse" für einen Dienst
  - ◆ Dienstbringer (Prozess) verbindet einen Socket mit dem Port
  - ◆ positive 16-bit Zahl
  - ◆ Ports bis einschließlich 1023 sind dem Superuser vorbehalten

## 4 Client-Server-Modell

- ★ Ein **Server** ist ein Programm, das einen **Dienst** (*Service*) anbietet, der über einen Kommunikationsmechanismus erreichbar ist (vgl. Vorl. 7-93, ungleichberechtigte Kommunikation)
- Server
  - ◆ **akzeptieren Anforderungen**, die von der Kommunikationsschnittstelle kommen
  - ◆ **führen** ihren angebotenen **Dienst aus**
  - ◆ **schicken** das **Ergebnis zurück** zum Sender der Anforderung
  - ◆ *Server* sind normalerweise als normale Benutzerprozesse realisiert
- Client
  - ◆ ein Programm wird ein **Client**, sobald es
    - eine **Anforderung an einen Server** schickt und
    - auf eine Antwort wartet

## 3 Socket-Typen

- **Stream-Sockets**
  - ◆ unterstützen bidirektionalen, zuverlässigen Datenfluss
  - ◆ gesicherte Kommunikation (gegen Verlust und Duplizierung von Daten)
  - ◆ die Reihenfolge der gesendeten Daten bleibt erhalten
  - ◆ Vergleichbar mit einer *pipe* – allerdings bidirektional (UNIX-Domain- und Internet-Domain-Sockets mit TCP/IP)
- **Datagramm-Sockets**
  - ◆ unterstützen bidirektionalen Datentransfer
  - ◆ Datentransfer unsicher (Verlust und Duplizierung möglich)
  - ◆ die Reihenfolge der ankommenden Datenpakete stimmt nicht sicher mit der der abgehenden Datenpakete überein
  - ◆ Grenzen von Datenpaketen bleiben im Gegensatz zu **Stream-Socket**-Verbindungen erhalten (Internet-Domain-Sockets mit UDP/IP)

## 5 Generieren eines Sockets

- Sockets werden mit dem Systemaufruf *socket(2)* angelegt

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

- **domain**, z. B.
  - ◆ **PF\_INET**: IPv4 (PF\_ = Protocol Family)
  - ◆ **PF\_INET6**: IPv6
- **type** innerhalb der Domain:
  - ◆ **SOCK\_STREAM**: Stream-Socket (bei PF\_INET(6) = TCP-Protokoll)
  - ◆ **SOCK\_DGRAM**: Datagramm-Socket (bei PF\_INET(6) = UDP-Protokoll)
- **protocol**
  - ◆ Default-Protokoll für Domain/Type Kombination: 0 (z.B. INET/STREAM -> TCP) (siehe **getprotobyname(3)**)

## 6 Namensgebung

- Sockets werden ohne Namen generiert
- der Systemaufruf **bind(2)** bindet einen Name an einen Socket

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

- ◆ **s**: socket
- ◆ **name**: Protokollspezifische Adresse  
Socket-Interface (`<sys/socket.h>`) ist zunächst protokoll-unabhängig

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

im Fall von **AF\_INET**: IP-Adresse / Port

- es wird konkret eine **struct sockaddr\_in** übergeben

- ◆ **namelen**: Länge der konkret übergebenen Struktur in Bytes

## 8 Namensgebung für IPv6-Sockets

- Name durch IP-Adresse und Port-Nummer definiert

```
struct sockaddr_in6 {
    uint16_t      sin6_family; /* = AF_INET6 */
    uint16_t      sin6_port;   /* Port */
    uint32_t      sin6_flowinfo;
    struct in6_addr sin6_addr; /* IPv6-Adresse */
    uint32_t      sin6_scope_id;
};
struct in6_addr {
    unsigned char s6_addr[16];
};
```

- ◆ **sin6\_addr**: IPv6-Adresse
  - **in6addr\_any** / **IN6ADDR\_ANY\_INIT**:  
auf allen lokalen Adressen Verbindungen akzeptieren

## 7 Namensgebung für IPv4-Sockets

- Name durch IP-Adresse und Port-Nummer definiert

```
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 */
};
```

- ◆ **sin\_port**: Port-Nummer
  - Port-Nummern sind eindeutig für einen Rechner und ein Protokoll
  - Port-Nummern < 1024: privilegierte Ports für root (in UNIX)  
(z.B. www=80, Mail=25, finger=79)
  - Portnummer = 0: die Portnummer soll vom System gewählt werden
  - Portnummern sind 16 Bit, d.h. kleiner als 65536
- ◆ **sin\_addr**: IP-Adresse
  - **INADDR\_ANY**: wenn Socket auf allen lokalen Adressen (z. B. allen Netzwerkinterfaces) Verbindungen akzeptieren soll

## 9 Binden eines TCP-Socket — Beispiel

- Adresse und Port müssen in Netzwerk-Byteorder vorliegen!
- Die **INADDR\_**-Werte liegen in Host-Byteorder vor
- Die **IN6ADDR\_**-Werte liegen bereits in Netzwerk-Byteorder vor
- Beispiel

```
#include <sys/types.h>
#include <netinet/in.h>
...
struct sockaddr_in6 sin;
...
s = socket(PF_INET6, SOCK_STREAM, 0);
sin.sin6_family = AF_INET6;
sin.sin6_addr = in6addr_any; /* bereits in NW-Byteorder */
sin.sin6_port = htons(MYPORT);
bind(s, (struct sockaddr *) &sin, sizeof(sin));
```

## 10 Verbindungsannahme durch Server

### ■ Server:

- ◆ **listen(2)** stellt ein, wie viele ankommende Verbindungswünsche gepuffert werden können (d.h. auf ein *accept* wartend)
- ◆ **accept(2)** nimmt Verbindung an:
  - *accept* blockiert solange, bis ein Verbindungswunsch ankommt
  - es wird ein neuer Socket erzeugt und an remote Adresse + Port (Parameter *from*) gebunden  
lokale Adresse + Port bleiben unverändert
  - dieser Socket wird für die Kommunikation benutzt
  - der ursprüngliche Socket kann für die Annahme weiterer Verbindungen genutzt werden

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

## 12 Verbindungsaufbau und Kommunikation

- ### ■ Beispiel: Server, der alle Eingaben wieder zurückschickt (ohne Fehlerbehandlungen)

```
fd = socket(PF_INET6, SOCK_STREAM, 0);

name.sin6_family = AF_INET6;
name.sin6_port = htons(port);
name.sin6_addr = ip6addr_any;

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

listen(fd, 5);

in_fd = accept(fd, NULL, NULL);

/* hier evtl. besser Kindprozess erzeugen und
   eigentliche Kommunikation dort abwickeln */
for(;;) {
    n = read(in_fd, buf, sizeof(buf));
    write(in_fd, buf, n);
}

close(in_fd);
close(fd);
```

## 11 Verbindungsaufbau durch Client

### ■ Client:

- ◆ **connect(2)** meldet Verbindungswunsch an Server
  - *connect* blockiert solange, bis Server Verbindung mit **accept** annimmt
  - Socket wird an die remote Adresse gebunden
  - Kommunikation erfolgt über den Socket
  - falls Socket noch nicht lokal gebunden ist, wird gleichzeitig eine lokale Bindung hergestellt (Port-Nummer wird vom System gewählt)

```
struct sockaddr_in6 server;
...
connect(s, (struct sockaddr *)&server, sizeof server);
```

- ### ■ Eine Verbindung ist eindeutig gekennzeichnet durch
- ◆ <lokale Adresse, Port> und <remote Adresse, Port>

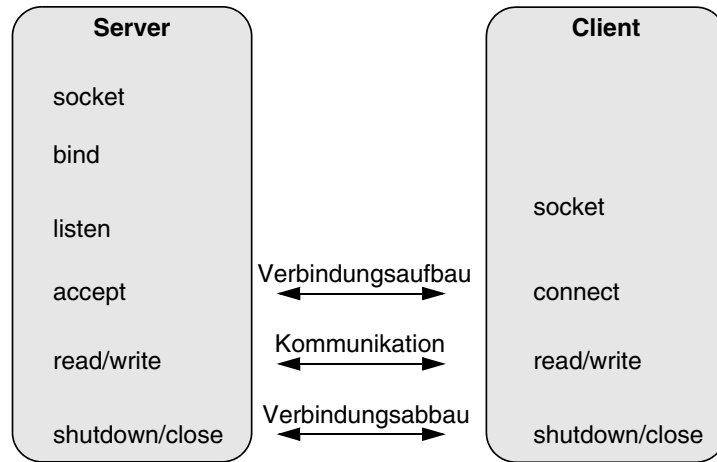
## 13 Schließen einer Socketverbindung

- ### ■ close(s)
- ### ■ shutdown(s, how)
- ◆ how:
    - SHUT\_RD: verbiete Empfang (nächstes *read* liefert EOF)
    - SHUT\_WR: verbiete Senden (nächstes *write* führt zu Signal SIGPIPE)
    - SHUT\_RDWR: verbiete Senden und Empfangen

## 14 Verbindungslose Sockets

- ### ■ Für Kommunikation über Datagramm-Sockets kein Verbindungsaufbau notwendig
- ### ■ Systemaufrufe
- |                    |                     |
|--------------------|---------------------|
| <b>sendto(2)</b>   | Datagramm senden    |
| <b>recvfrom(2)</b> | Datagramm empfangen |
- ### ■ Besonderheit: **Broadcasts** über Datagramm-Sockets (Internet Domain)

## 15 TCP-Sockets: Zusammenfassung



## 16 Sockets und UNIX-Standards

- Sockets sind nicht Bestandteil des POSIX.1-Standards
- Sockets stammen aus dem BSD-UNIX-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)

## U9-4 Netzwerk-Programmierung - Verschiedenes

- Parametrierung eines Sockets abfragen / setzen
  - ◆ **getsockopt(2), setsockopt(2)**
- Informationen über Socket-Bindung
  - ◆ **getpeername(2)**  
Namen der mit dem Socket verbundenen Gegenstelle abfragen
  - ◆ **getsockname(2)**  
Namen eines Sockets abfragen
- Hostnamen und -adressen ermitteln
  - ◆ **getnameinfo(3)**: Reverse-Lookup: IP-Adresse => DNS-Name
  - ◆ **getaddrinfo(3)**: Forward-Lookup: DNS-Name => IP-Adressen

## 1 Socket-Optionen setzen

```
#include <sys/socket.h>
int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);
```

- Verfügbare Optionen abhängig von Protokollfamilie (**ip(7)**, **ipv6(7)**)
- Beispiel (ohne Fehlerbehandlungen):

```
int sock, sopt_value;

sock = socket(PF_INET6, SOCK_STREAM, 0);

sopt_value = 1;

/* Sofortiges Neubinden eines kürzlich gebundenen Socketnamens zulassen */
setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &sopt_value, sizeof(int));

/* IPv6-Socket nicht für IPv4-mapped-Verbindungen verwenden */
setsockopt(sock, IPPROTO_IPV6, IPV6_V6ONLY, &sopt_value, sizeof(int));
```

## 2 getsockname, getpeername

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

### ■ Informationen über die lokale Adresse des Socket

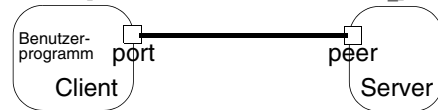
```
struct sockaddr_in6 server;
socklen_t len;

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_in6 server;
socklen_t len;

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



## 3 DNS-Forward-Lookup

### ■ getaddrinfo liefert Socket-Namen zu einem Host/Dienst-Paar

```
int getaddrinfo(const char *node, const char *service,
               const struct addrinfo *hints, struct addrinfo **res);
```

### ■ node gibt den DNS-Namen des Hosts an (oder IP-Adresse als String)

### ■ service gibt entweder numerischen Port als String (z.B. "25") oder den Dienstnamen (z.B. "smtp", **getservbyname(3)**) an

### ■ Mit hints kann die Adressauswahl eingeschränkt werden (z.B. auf IPv4-Sockets). Nicht verwendete Felder auf 0 bzw. NULL setzen.

### ■ Ergebnis ist eine verkettete Liste von Socket-Namen; ein Zeiger auf das Kopfelement wird in res gespeichert

### ■ Fehlerbehandlung siehe **getaddrinfo(3)**

### ■ Freigabe der Ergebnisliste nach Verwendung mit **freeaddrinfo(3)**

## 3 DNS-Forward-Lookup

```
struct addrinfo {
    int          ai_flags; /* flags zur Auswahl (hints) */
    int          ai_family; /* z.B. PF_INET6 */
    int          ai_socktype; /* z.B. SOCK_STREAM */
    int          ai_protocol; /* Protokollnummer */
    size_t       ai_addrlen; /* Größe von ai_addr */
    struct sockaddr *ai_addr; /* Adresse f. bind/connect */
    char         *ai_canonname; /* offizieller Hostname */
    struct addrinfo *ai_next; /* nächste Adresse oder NULL */
};
```

### ■ ai\_flags relevant zur Anfrage von Auswahlkriterien (hints)

- ◆ **AI\_ADDRCONFIG**: Auswahl von Adresstypen, für die auch ein lokales Interface existiert (z.B. werden keine IPv6-Adressen geliefert, wenn der aktuelle Rechner gar keine IPv6-Adresse hat)

### ■ ai\_family, ai\_socktype, ai\_protocol für **socket(2)** verwendbar

### ■ ai\_addr, ai\_addrlen als für **bind(2)** und **connect(2)** verwendbar

## 4 Beispiel: Verbindung aufbauen

```
char *hostname = "fau40.informatik.uni-erlangen.de";
int gai_ret, sock;
struct addrinfo *sa_head, *sa, hints;

memset(&hints, 0, sizeof(hints));
hints.ai_socktype = SOCK_STREAM; /* nur TCP-Sockets */
hints.ai_family = PF_UNSPEC; /* beliebige Protokollfamilie */
hints.ai_flags = AI_ADDRCONFIG; /* nur lokal verf. Adresstypen */

gai_ret = getaddrinfo(hostname, "25", &hints, &sa_head);
if(gai_ret != 0) { /* Fehlerbehandlung s. Manpage */ }

/* Liste der Adressen durchtesten */
for(sa = sa_head; sa != NULL; sa = sa->ai_next) {
    sock = socket(sa->ai_family, sa->ai_socktype, sa->ai_protocol);
    if(0 == connect(sock, sa->ai_addr, sa->ai_addrlen)) {
        break;
    }
    close(sock);
}
if(sa == NULL) { /* Fehler */ }

freeaddrinfo(sa_head);
```

## U9-5 Transparente IPv4-in-IPv6-Unterstützung

- Spezieller Adressbereich ::ffff:0:0/96 zur Abbildung von IPv4 auf IPv6
- z.B. 131.188.30.102 auf ::ffff:83bc:1e66 (auch ::ffff:131.188.30.102)
- Binden eines PF\_INET6-Sockets bindet standardmäßig auch den entsprechenden IPv4-Port
  - ◆ dem Prozess erscheinen eingehende IPv4-Verbindungen als IPv6-Verbindungen aus diesem Adressbereich
- ausgehende IPv6-Verbindungen an diesen Adressbereich werden auf entsprechende IPv4-Verbindungen abgebildet

## U9-6 Wichtige Socket-Manpages

- Socket erzeugen: **socket(2)**
- Client: **connect(2)**
- Server: **bind(2), listen(2), accept(2)**
- IP-Protokolle, sockaddr-Strukturen: **ip(7), ipv6(7)**
- DNS: **getaddrinfo(3), getnameinfo(3)**
- Socket-Optionen: **setsockopt(2)**