FH | JOANNEUM
University of Applied Sciences

Systemnahe Programmierung
SS 2024

Unit 8

Netzwerkprogrammierung

Sockets

Sockets sind Kommunikationsendpunkte für Netzwerkverbindungen.

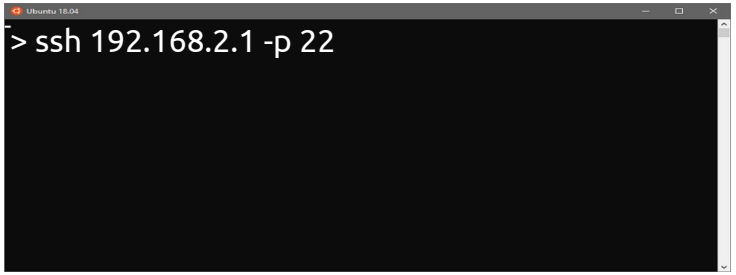
- Auf beiden Seiten des Kommunikationskanals werden Sockets erzeugt.
- Sockets können analog zu Low-Level Filedeskriptoren verwendet werden.
- Socketverbindungen sind bidirektional.
- Es wird zwischen Client und Server unterschieden, Server akzeptieren eingehende Verbindungen, Clients initiieren den Verbindungsaufbau.

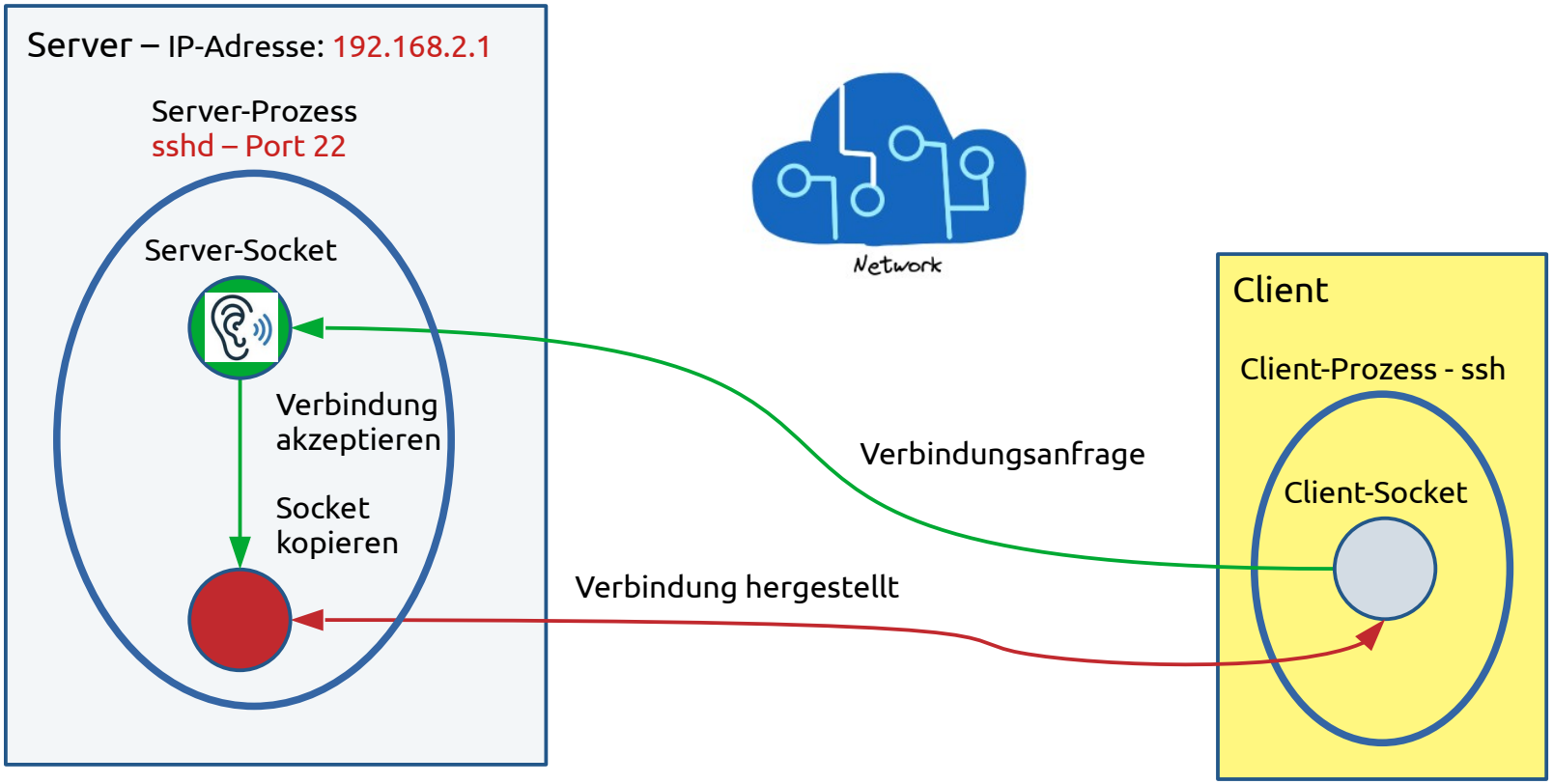
Netzwerkprogrammierung

Sockets – Beispiel ssh Verbindung

Beispiel: ssh 192.168.2.1 -p 22

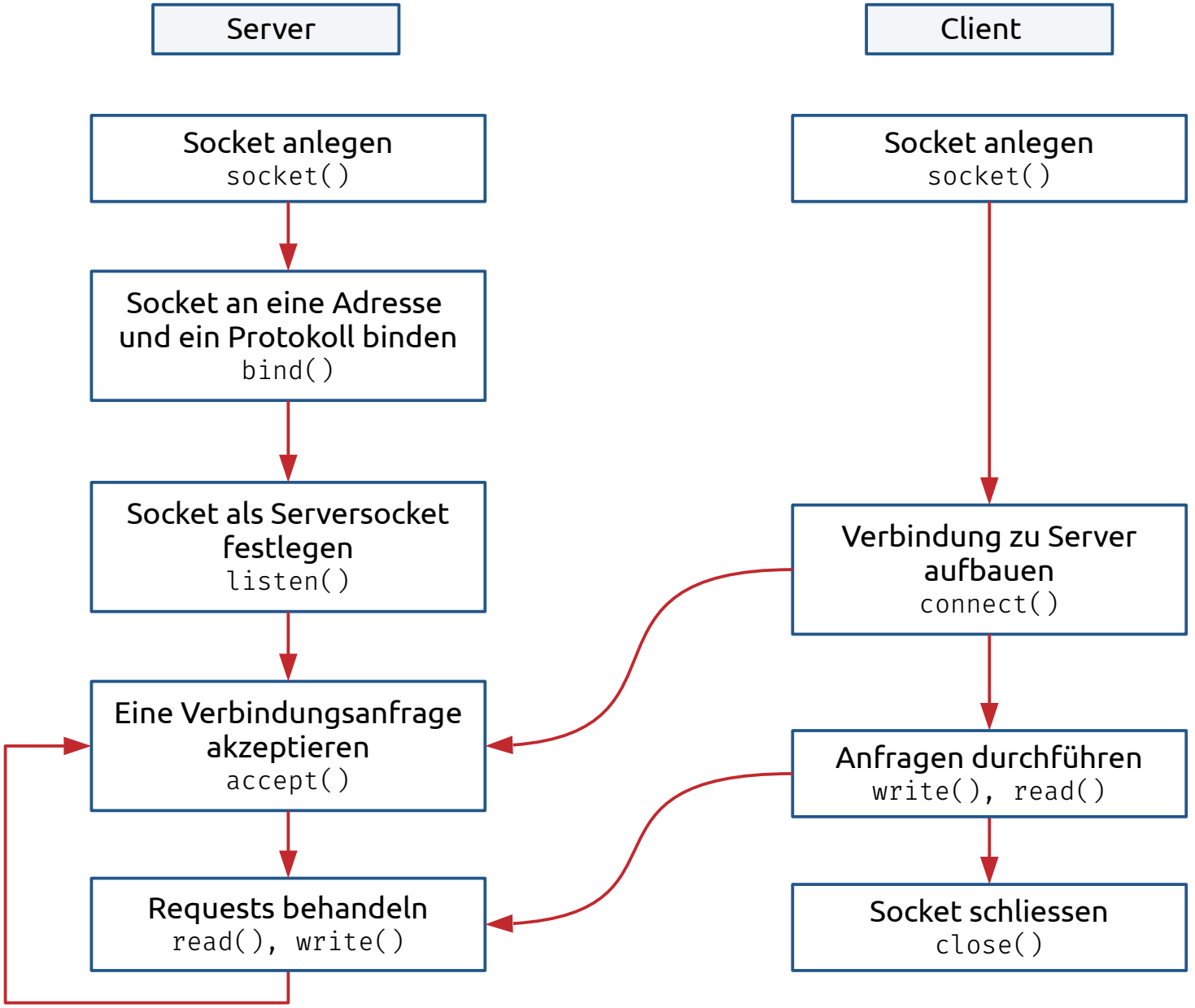
Adressierung:
- Protokoll IPv4
- IP-Adresse – 192.168.2.1
- Port - 22





Netzwerkprogrammierung

Client und Server - Verbindungsaufbau



Netzwerkprogrammierung

Sockets anlegen

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

Family - Protokoll-Familie

`AF_UNIX` ... Unix Socket - für lokale Kommunikation auf einem Host
`AF_INET` ... Internet Socket
`AF_INET6` ... Internet V6 Socket

Type

`SOCK_STREAM` ... TCP
`SOCK_DGRAM` ... UDP
`SOCK_RAW` ... Raw IP

Protocol
Falls innerhalb der `family` ein Subprotokoll zu unterscheiden ist, kann es mit diesem Parameter ausgewählt werden. Normalerweise 0.

Netzwerkprogrammierung

Sockets binden

`bind()` bindet einen Socket an eine spezifische IP-Adresse (Netzwerkinterface) und einen Port.

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Für den Parameter `addr` kann für Internetprotokolle mit `sockaddr_in` verwendet werden. Er beschreibt die Adressparameter einer Netzwerkverbindung - Protokoll, IP-Adresse und Port.

```
struct sockaddr_in {
  unsigned short int  sin_family; /* AF_INET     */
  unsigned short int  sin_port;   /* Portnummer */
  struct in_addr      sin_addr;   /* Netzwerkinterface oder INADDR_ANY  */
};
```

Netzwerkprogrammierung

Hilfsfunktionen

Manche Werte müssen zwischen der sog. **„Network Byte Order** in die **„Host Byte Order“** und umgekehrt konvertiert werden.

> *"On the i386 the host byte order is Least Significant Byte first (Little-Endian), whereas the network byte order, as used on the Internet, is Most Significant Byte first (Big-Endian)."*

Konvertieren von Hostorder in Networkorder:

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
```

Konvertieren von Networkorder in Hostorder:

```
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

Netzwerkprogrammierung

Den Socket als Serversocket festlegen

`listen()` markiert den Socket als passiv, d.h. als Serversocket, der eingehende Verbindungen akzeptiert

```
#include <sys/types.h>
#include <sys/socket.h>

int listen(int sockfd, int backlog);
```

Der Parameter `backlog` legt die Länge der Queue für wartende Verbindungen fest. Im Normalfall auf `SOMAXCONN` setzen.

Netzwerkprogrammierung

Auf eingehende Verbindungen warten

`accept()` wartet auf eingehende Verbindungen.

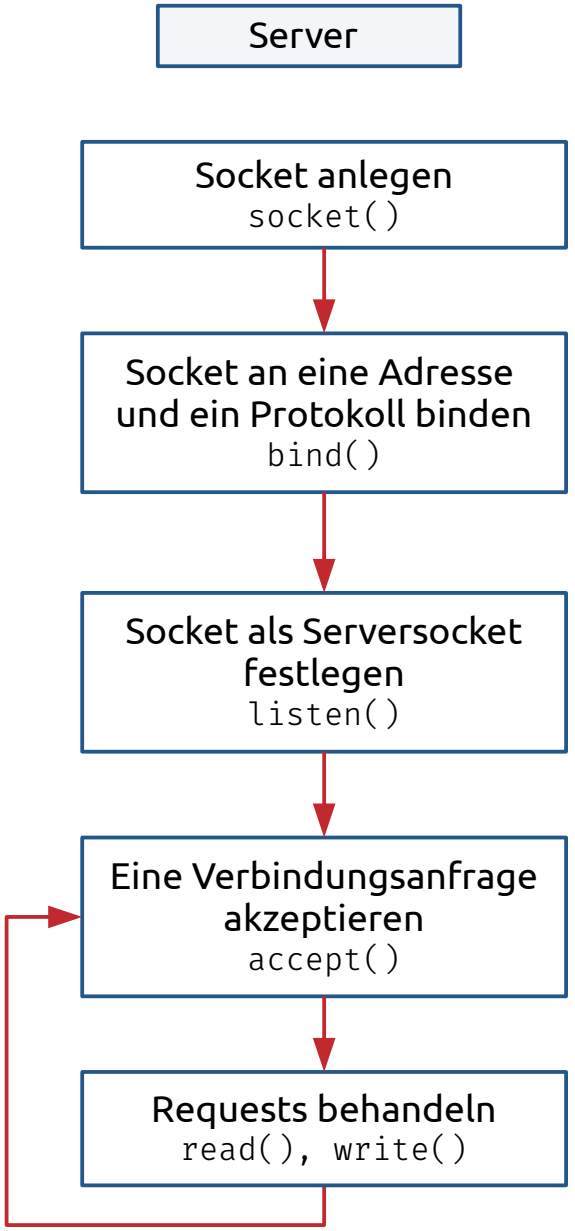
```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Der Parameter `sockfd` bezeichnet den Server-Socket. Wenn eine eingehende Verbindung akzeptiert wird, dann liefert `accept()` einen neuen Socket-Descriptor für die Clientverbindung zurück.

Der Parameter `addr` liefert die Verbindungsparameter des Clients und ist vom Typ `struct sockaddr_in`.

Netzwerkprogrammierung

Server – Verbindungen bearbeiten




```
struct sockaddr_in serveraddr;
struct sockaddr_in clientaddr;
int clientAddrLen = sizeof(clientaddr);
int sockfd, newsockfd;

sockfd = socket(AF_INET, SOCK_STREAM, 0)

bzero((char *) &serveraddr, sizeof(struct sockaddr_in));
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
serveraddr.sin_port = htons(17000);

bind(sockfd,(struct sockaddr *)&serveraddr, sizeof(struct sockaddr_in));

listen(sockfd, SOMAXCONN);

newsockfd = accept(sockfd,(struct sockaddr *) &clientaddr, (socklen_t *)&clientAddrLen);

msize = read(newsockfd, buffer, MAX_MSG_SIZE);

status = write(newsockfd,mesg,strlen(mesg));
```

Netzwerkprogrammierung

Beispiel Server

```
struct sockaddr_in serveraddr;
struct sockaddr_in clientaddr;
int sockfd, newsockfd;

if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) { → Socket anlegen
...
}
bzero((char *) &serveraddr, sizeof(struct sockaddr_in)); → nicht benötigte Felder löschen
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
serveraddr.sin_port = htons(17000); → in „network byte order“ konvertieren

if (bind(sockfd,(struct sockaddr *)&serveraddr, sizeof(struct sockaddr_in)) < 0) {
     → an Port 1700 binden
...
}

listen(sockfd, SOMAXCONN); → Serversocket
```

Netzwerkprogrammierung

Beispiel Server

```
if ((newsockfd = accept(sockfd,(struct sockaddr *) &clientaddr,
        (socklen_t *)&clientAddrLen)) < 0) { → auf eingehende Verbindungen warten
  if ( errno == ECONNABORTED ) { → abgebrochene Verbindungen ignorieren
    continue;
  }
  else if ( errno == EINTR) { → Unterbrechung durch Signal
    continue;
  }
  else {
     ... Error ...
  }
}

while ((bytesRead = read(newsockfd, buffer, BUFFER_SIZE)) > 0) {  → Lesen
   ..
}
```

Netzwerkprogrammierung

Übung 1

Schreiben sie ein Programm `server.c`, das einen Socket-Server, der auf Port `17000` Verbindungen akzeptiert, implementiert.

Lesen sie den Netzwerkstream einmal mit `read()` und geben sie die empfangenen Zeichen auf `stdout` aus.
Nehmen sie für die Größe des Read-Buffers den Wert `256` an.

Nachdem die Daten gelesen wurden, senden sie mit `write()` folgenden String an den Absender zurück

```
"HTTP/1.1 200 OK\r\n\r\nHallo!\r\n"
```

und schließen dann den Socket.

Klammern sie `accept()`, `read()` und `write()` in einer Endlosschleife, so dass neue Verbindungen aufgebaut werden können sobald die Aktuelle beendet ist.

Zum Testen können sie folgenden Befehl verwenden:

```
curl --get http://localhost:17000
```

Netzwerkprogrammierung

Socket Optionen

Mit diesen Funktionen kann das Verhalten eines Sockets festgelegt/abgefragt werden.

```
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(int sockfd, int level, int optname,
               void *optval, socklen_t *optlen);

int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen);
```

Netzwerkprogrammierung

Socket Optionen

Level beschreibt die Schicht in der die Optionen gesetzt/abgefragt werden sollen:
`SOL_SOCKET`, `IPPROTO`, `IPPROTO_ICMPV6`, `IPPROTO_IPV6`, `IPROTO_TCP`

Der Parameter `optname` enthalt den Namen der gewünschten Option.

Diese werden wir verwenden:
`SO_REUSEADDR` ... erlaubt die Verwendung eine Adresse auch wenn sie noch in Verwendung ist (Status TIMEWAIT)

`SO_KEEPALIVE` ... prüft in regelmässigen Abständen ob die Verbindung noch aufrecht ist.
Weitere Optionen → manpage!

Der Wert von `optval` ist 1 wenn die Option gesetzt ist und sonst 0.

Netzwerkprogrammierung

Socket Optionen

Beispiel

```
int opt=1;
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (int *) &opt, sizeof opt);

int opt=1;
setsockopt(sockfd, SOL_SOCKET, SO_KEEPALIVE, (int *) &opt, sizeof opt);
```

Netzwerkprogrammierung

Mehrere Verbindungen parallel aufbauen

Unser Webserver soll natürlich in der Lage sein, mit mehreren Clients gleichzeitig zu kommunizieren.

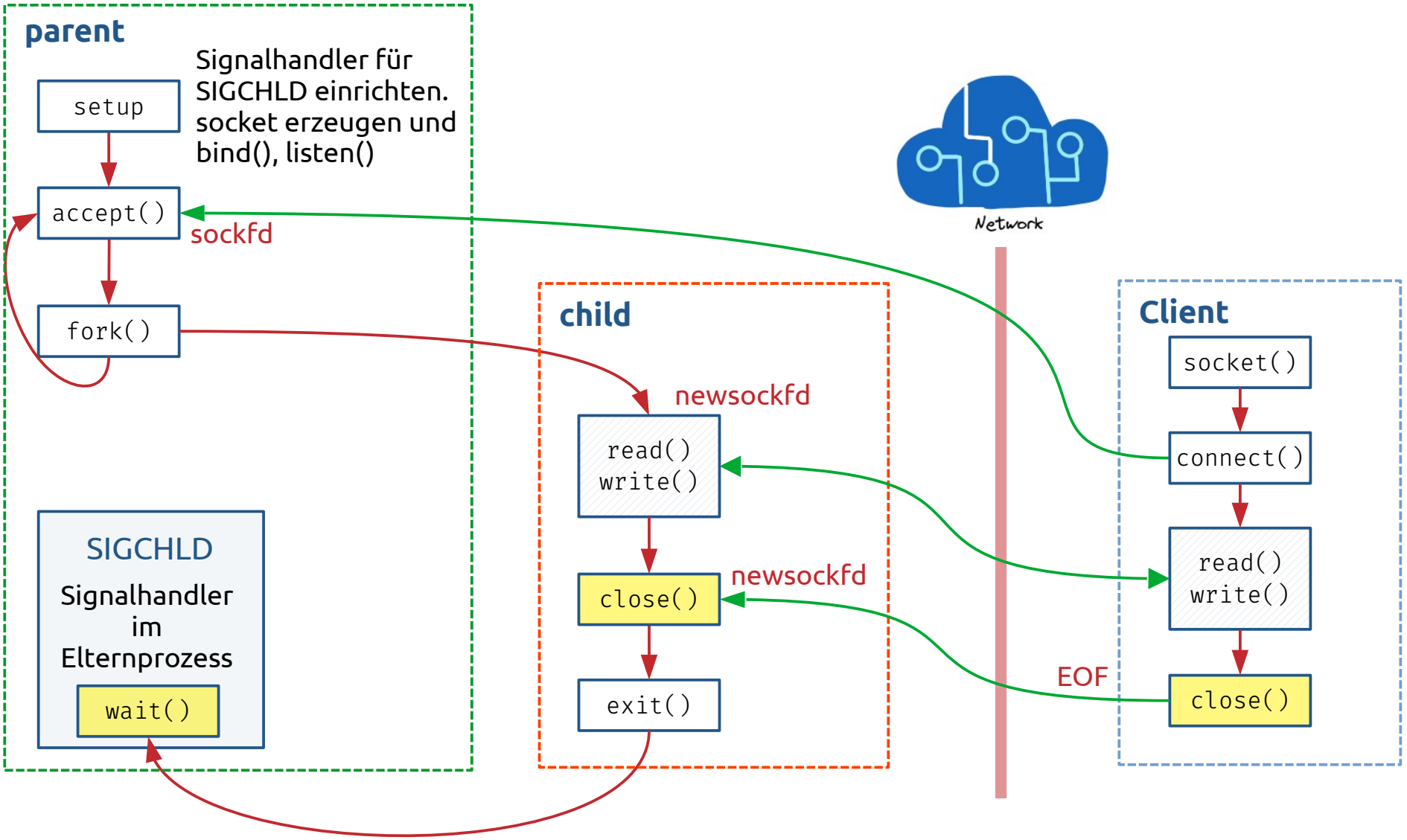
`accept()` baut die Verbindung auf, da wir aber anschließend den Socket exklusiv verwenden, ist kein weiterer Verbindungsaufbau möglich, solange die bestehende Verbindung existiert.

Daher müssen wir mit `fork()` einen Kindprozess starten, der die Client-Verbindung übernimmt und der Elternprozess kann dann weitere Verbindungsanfragen behandeln.

Da der Elternprozess mit `accept()` blockiert ist, muss für die saubere Beendigung des Kindprozesses das Signal `SIGCHLD` abgefangen werden und im Signalhandler dann `wait()` aufgerufen werden.

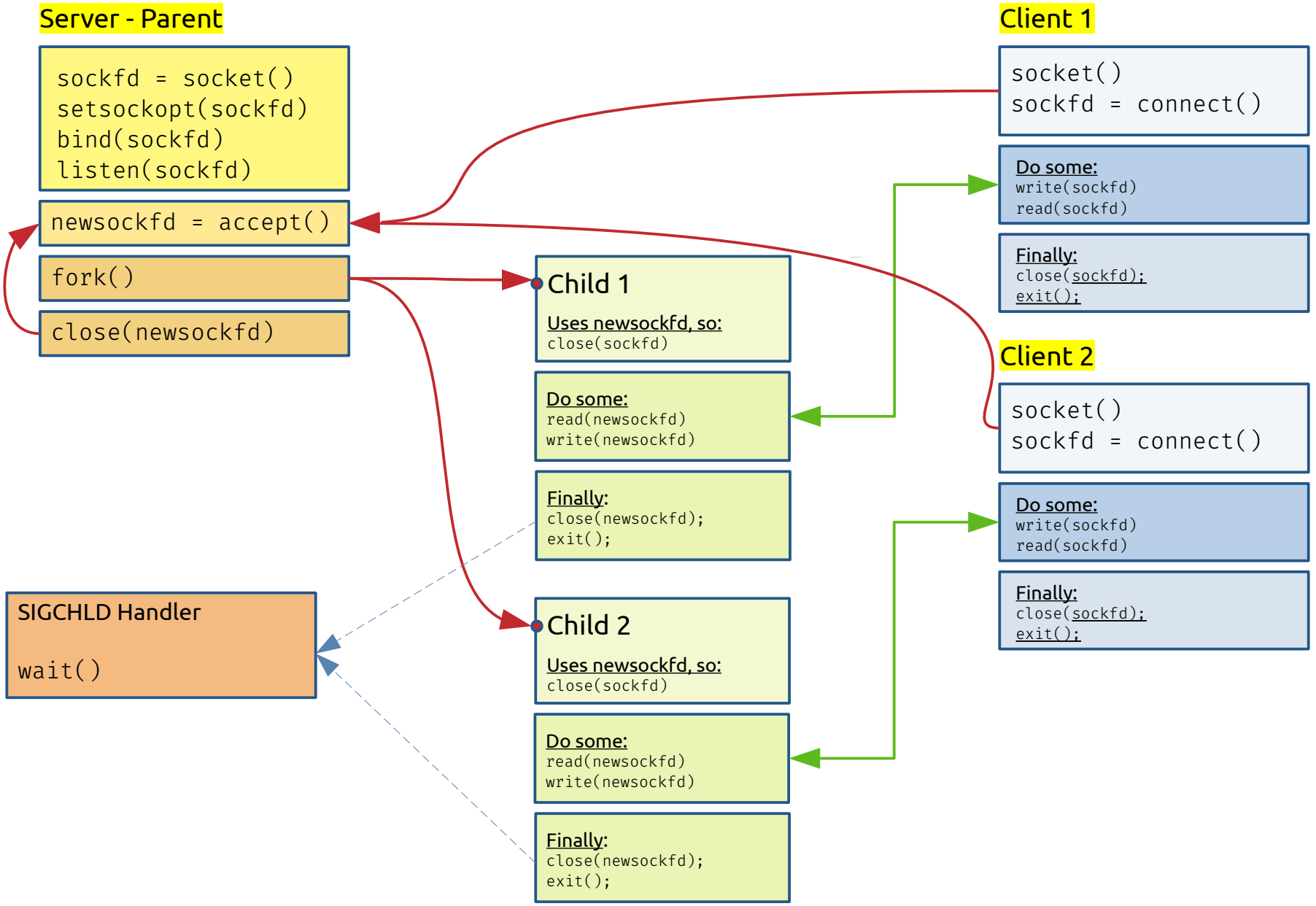
Netzwerkprogrammierung

Mehrere Verbindungen akzeptieren



Netzwerkprogrammierung

Mehrere Verbindungen akzeptieren



Netzwerkprogrammierung

Socket-Verbindung vom Client

Mit `connect()` kann der Client eine Verbindung zum Server aufbauen. Ein Socket muss vorher analog wie im Server erzeugt werden.

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr,
                    socklen_t addrlen);
```

Parameter:

`sockfd` ... Socket Deskriptor

`addr` ... Struktur mit der Verbindungsinformation → wir verwenden wieder die Struktur mit dem Namen `sockaddr_in` für das Internetprotokoll TCP.

In der Struktur `sockaddr_in` müssen folgende Felder gesetzt werden:

`sin_family` ... Protokollfamilie, in unserem Fall `AF_INET`

`sin_addr` ... IP-Adresse des Servers in binärer Form

`sin_port` ... Server-Port in Networkorder

Netzwerkprogrammierung

IP-Adressen umwandeln

IP-Adressen in Text Form („xxx.xxx.xxx.xxx“) müssen in eine binäre und der Networkorder entsprechende Form umgewandelt werden.

```
#include <netinet/in.h>
#include <arpa/inet.h>
```

Text Form in binäre Form umwandeln:

```
int inet_aton(const char *cp, struct in_addr *inp);
```

`cp` ... Internetadresse in String Form (z.B.: 127.0.0.1)
`inp` ... `sin_addr` Komponente der `sockaddr_in` Struktur

Binäre Form in einen String konvertieren:

```
const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
```

`af` ... Protokollfamilie = `AF_INET`
`src` ... `sin_addr` Komponente der `sockaddr_in` Struktur
`dst` ... Stringbuffer für das Ergebnis der Konvertierung
`size` ... Länge des Stringbuffers

Netzwerkprogrammierung

Client Socket-Verbindung

Die Struktur `sockaddr_in` für die Clientverbindung verwenden:

```
struct sockaddr_in serveraddr;

/* die Struktur löschen */
bzero((char *) &serveraddr, sizeof(struct sockaddr_in));

/* Family und Port setzen */
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(PORTNUMBER); // bei uns 17000

/* IP-Adresse in binäre Form umwandeln */
inet_aton ("127.0.0.1", &serveraddr.sin_addr); // localhost
```

Netzwerkprogrammierung

Beispiel Client

```
char serverIP[20] = "127.0.0.1";
int sockfd=0;
struct sockaddr_in serveraddr;

/* set server address information */
bzero((char *) &serveraddr, sizeof(struct sockaddr_in));
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(PORTNUMBER);
inet_aton(serverIP, &serveraddr.sin_addr);

/* create socket */
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
  perror("socket");
  exit(EXIT_FAILURE);
}

/* Now connect to the server */
if (connect(sockfd, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) < 0) {
  perror("connect");
  exit(EXIT_FAILURE);
}
```

Netzwerkprogrammierung

Übung

Erweitern sie ihren Server so, dass er mehrere Verbindungen gleichzeitig aufbauen und bearbeiten kann.

Verwenden sie `fork()` um die akzeptierte Verbindung in einem Kindprozess abzuarbeiten.

Installieren sie einen Signalhandler für das Signal `SIGCHLD` und warten sie im Handler auf den Kindprozess. Geben sie in diesem Signalhandler die Meldung `"Waited for child"` aus, damit sichtbar wird wann der Prozess beendet wurde.

Ignorieren sie das Signal `SIGPIPE`, damit, wenn ein Client beendet wird, nicht der Serverprozess terminiert wird.

Setzen sie für den Server-Socket die Socket-Optionen `SO_REUSEADDR` und `SO_KEEPALIVE`.

Netzwerkprogrammierung

Übung

Geben nach jedem `fork()` die IP Adresse des Clients (`inet_ntop`) und die PID des neu erzeugten Child-Prozesses in folgender Form aus:

```
Started new process 23014 for client: 127.0.0.1
```

Testen

Um mehrere Verbindungen zu testen, bauen sie (in einem separaten Terminal) eine zusätzliche Verbindung zum Server mit telnet auf

```
telnet localhost 17000
```

und lassen diese Verbindung so stehen.

Testen sie nun wie gehabt mit `curl` ob mehrere Verbindungen gleichzeitig möglich sind.