- Ein Netzwerkdienst ist ein Prozess, der unter einer Netzwerkadresse einen Dienst anbietet.
- Ein Klient, der die Netzwerkadresse kennt, kann einen bidirektionalen Kommunikationskanal zu dem Netzwerkdienst eröffnen und über diesen mit dem Dienst kommunizieren.
- Die Kommunikation wird durch ein Protokoll strukturiert, bei dem typischerweise Anfragen oder Kommandos auf dem Hinweg übermittelt werden und auf dem Rückweg des Kommunikationskanals die zugehörigen Antworten kommen.
- Wenn erst die Antwort gelesen werden muss, bevor die nächste Anfrage gestellt werden darf, wird von einem *synchronen* Protokoll gesprochen.
- Wenn mehrere Anfragen unmittelbar hintereinander gestellt werden dürfen, ohne dass erst die Antworten abgewartet werden, wird von *Pipelining* gesprochen. (Das hat nichts mit den Pipes aus dem vorherigen Kapitel zu tun.)

- Die beiden Kommunikationspartner müssen nicht miteinander verwandt sein.
- Sie müssen nicht einmal auf dem gleichen Rechner laufen.
- Da der Kommunikationskanal bidirektional ist, wird ein echter Dialog zwischen den beiden Prozessen möglich.
- Der Aufbau einer Verbindung ist jedoch schwieriger, da zunächst die Netzwerkadresse des gewünschten Partners ermittelt werden muss.

Wenn Dienste über das Netzwerk angeboten und in Anspruch genommen werden, ergeben sich viele Vorteile:

- Der Dienst kann allen offenstehen, und ein direkter Zugang zu dem Rechner, auf dem der Dienst angeboten wird, ist nicht notwendig.
- Viele Parteien können in kooperativer Weise einen Dienst gleichzeitig nutzen.
- Der Dienste-Anbieter hat weniger Last, da die Benutzerschnittstelle auf anderen Rechnern laufen kann.

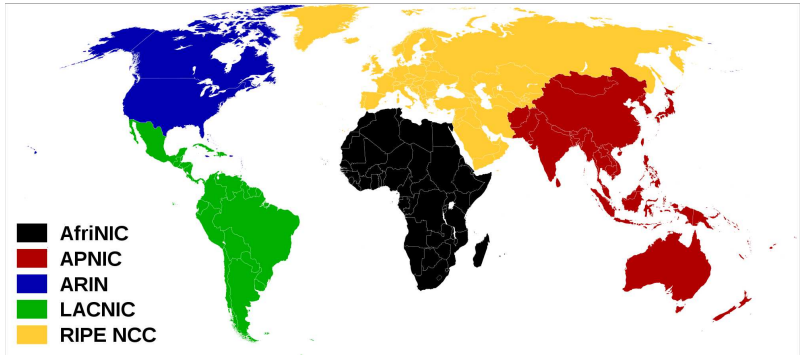
- Der Kreis derjenigen, die auf einen Netzwerkdienst zugreifen können, ist möglicherweise ziemlich umfangreich (normalerweise das gesamte Internet).
- Somit muss jeder Netzwerkdienst Zugriffsberechtigungen einführen und überprüfen und kann sich dabei nicht wie traditionelle Applikationen auf die des Betriebssystems verlassen.
- Dienste, die gleichzeitig von vielen genutzt werden können, haben vielerlei zusätzliche Konsistenz- und Synchronisierungsprobleme, für die nicht jede Art von Datenhaltung geeignet ist.
- Netzwerke bringen neue Arten von Ausfällen mit sich, wenn eine Netzwerkverbindung zusammenbricht oder es zu längeren „Hängern“ kommt.

- Im Rahmen dieser Vorlesung beschäftigen wir uns vorwiegend mit TCP/IP, also dem verbindungsorientierten Protokoll des Internets. (Mehr zur Semantik später.)
- Im Internet gibt es zwei etablierte Räume für Netzwerkadressen: IPv4 und IPv6.
- IPv4 arbeitet mit 32-Bit-Adressen und ist seit dem 1. Januar 1983 in Benutzung.
- Da der Adressraum bei IPv4 auszugehen droht, gibt es als Alternative IPv6, das mit 128-Bit-Adressen arbeitet.
- Im Rahmen dieser Vorlesung beschäftigen wir uns nur mit IPv4.
- Eine IPv4-Adresse (das gilt auch für IPv6) adressiert nur den Rechner, auf dem der Dienst läuft. Der Dienst selbst wird über eine Portnummer (16 Bit) ausgewählt.
- Ein Netzwerkdienst wird also z.B. über eine IPv4-Adresse und eine Port-Nummer adressiert.

```
clonard$ telnet 134.60.54.12 13
Trying 134.60.54.12...
Connected to 134.60.54.12.
Escape character is '^]'.
Mon Jun 14 11:03:16 2010
Connection to 134.60.54.12 closed by foreign host.
clonard$
```

- 134.60.54.12 ist eine IPv4-Adresse in der sogenannten *dotted-decimal*-Notation, bei der durch Punkte getrennt jedes der vier Bytes der Adresse einzeln dezimal spezifiziert wird.
- 134.60.54.12 ist also eine lesbarere Form für 2252092940.
- 13 ist die Port-Nummer des *daytime*-Dienstes.
- Die Port-Nummer ist nicht zufällig. Die 13 ist explizit von der IANA (*Internet Assigned Numbers Authority*) dem *daytime*-Dienst zugewiesen worden.

Aufteilung des Netzwerkraums



- Die IANA teilt den globalen IPv4-Adressraum auf einzelne lokale Institutionen, den sogenannten *Regional Internet Registries*.
- ARIN ist zuständig für Amerika, RIPE für Europa, den Mittleren Osten und Zentralasien, APNIC für Asien, Australien und Ozeanien, AfriNIC für Afrika und LACNIC für Lateinamerika einschließlich Teile der Karibik.
- Die Universität Ulm hat seit 1989 den Adressbereich *134.60.0.0/16*.

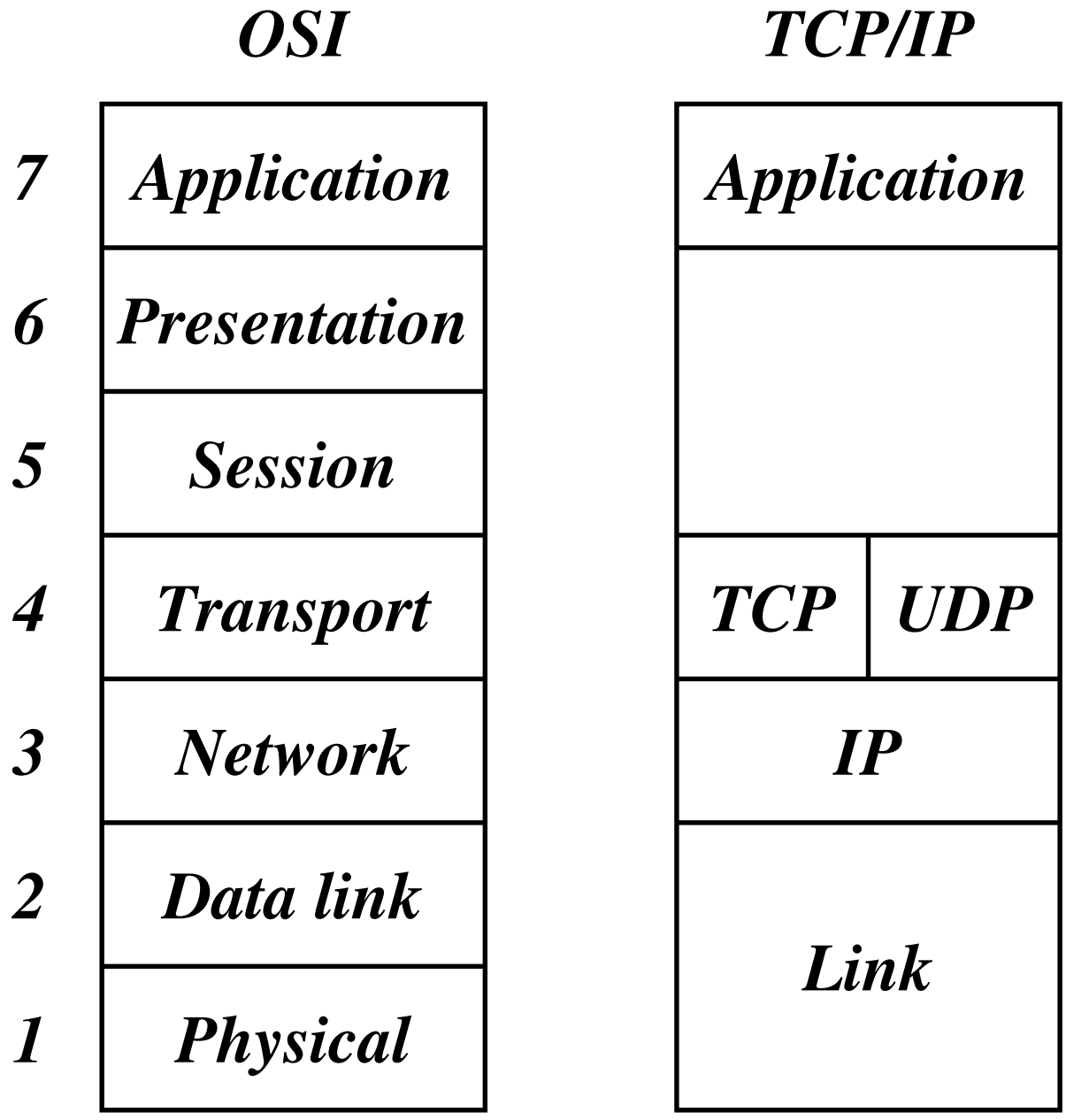
Wer ist zuständig für die 134.60.54.12? 182

```
thales$ wget -O - -q http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.txt | \
> sed 's/  */ /g' | grep '^ 134'
 134/8 Administered by ARIN 1993-05 whois.arin.net https://rdap.arin.net/registry LEGACY
thales$ whois -h whois.arin.net 134.60.66.5
[...]
NetRange:       134.58.0.0 - 134.61.255.255
CIDR:           134.60.0.0/15, 134.58.0.0/15
NetName:        RIPE-ERX-134-58-0-0
NetHandle:      NET-134-58-0-0-1
Parent:         NET134 (NET-134-0-0-0-0)
NetType:        Early Registrations, Transferred to RIPE NCC
OriginAS:
Organization:   RIPE Network Coordination Centre (RIPE)
RegDate:        2003-11-26
Updated:        2003-11-26
Comment:        These addresses have been further assigned to users in
Comment:        the RIPE NCC region.  Contact information can be found in
Comment:        the RIPE database at http://www.ripe.net/whois
[...]
thales$ whois -h whois.ripe.net 134.60.66.5
[...]
inetnum:        134.60.0.0 - 134.60.255.255
netname:        UNI-ULM
descr:          Ulm, Germany
country:        DE
[...]
route:          134.60.0.0/16
descr:          UNI-ULM
origin:         AS553
[...]
```

- Für Rechnernamen wie *thales.mathematik.uni-ulm.de* können über hierarchisierte Domain-Server die zugehörigen IP-Adressen abgefragt werden.
- Die Abfrage beginnt zuerst bei einem der 13 sogenannten Root-Server, die weltweit verteilt sind und deren IP-Adressen jedem Domain-Server bekannt sind.
- Einer davon ist *198.41.0.4*. Dieser verrät, welche Nameserver für die Top-Level-Domain *de* zuständig ist.
- Einer davon ist *194.0.0.53*. Dieser verrät, welche Nameserver für *uni-ulm.de* zuständig sind.
- Einer davon ist *134.60.1.111*, der sogleich in der Lage ist, diesen Namen vollständig aufzulösen und die *134.60.66.5* zurückzuliefern.

- IP-Adressen wie *134.60.66.5* werden nur auf einer abstrakten Ebene zur Verfügung gestellt.
- IP-Adressen werden auf der darunterliegenen physischen Ebene und denen damit verbundenen Protokollen nicht verstanden.
- So wird beispielsweise beim Ethernet, das bei uns weitgehend an der Universität zum Einsatz kommt, mit 6-Byte-Adressen gearbeitet.
- Die Thales hat beispielsweise die Ethernet-Adresse *00:19:99:ba:45:78* (Bytes werden hier in Form von Hexzahlen angegeben). Diese Adressen sind jedoch nur lokal auf einem Ethernet-Segment von Bedeutung.

- Aufbauend auf der Schicht mit IP-Adressen (IP-Protokoll) gibt es alternative Transport-Schichten, über die Pakete versendet werden können.
- Mittels UDP (*User Datagram Protocol*) können einzelne Pakete sehr effizient, aber unzuverlässig versendet werden.
- Im Gegensatz dazu gewährleistet TCP (*Transmission Control Protocol*) eine sichere Verbindung, die jedoch mehr Verwaltungsaufwand mit sich bringt.
- Parallel zu TCP/IP entstand 1983 das OSI-Referenz-Modell (*Open Systems Interconnection*), das eine feinere Schichtung vorsieht. Die Präsentations- oder Sitzungsebene fand jedoch nie ihren Weg in die Protokollhierarchie von TCP/IP.


OSI
TCP/IP
7
Application
6
Presentation
5
Session
4
Transport
3
Network
2
Data link
1
Physical
Application
TCP
UDP
IP
Link

- Für TCP/IP gibt es zwei Schnittstellen, die beide zum POSIX-Standard gehören:
- Die Berkeley Sockets wurden 1983 im Rahmen von BSD 4.2 eingeführt. Dies war die erste TCP/IP-Implementierung.
- Im Jahr 1987 kam durch UNIX System V Release 3.0 noch TLI (*Transport Layer Interface*) hinzu, die auf Streams basiert (einer anderen System-V-spezifischen Abstraktion).
- Die Berkeley-Socket-Schnittstelle hat sich weitgehend durchgesetzt. Wir werden uns daher nur mit dieser beschäftigen.

Die Entwickler der Berkeley-Sockets setzten sich folgende Ziele:

- **Transparenz**: Die Kommunikation zwischen zwei Prozessen soll nicht davon abhängen, ob sie auf dem gleichen Rechner laufen oder nicht.
- **Effizienz**: Zu Zeiten von BSD 4.2 (also 1983) war dies ein außerordentlich wichtiges Kriterium wegen der damals noch sehr geringen Rechenleistung. Aus diesem Grund werden insbesondere keine weiteren System-Prozesse zur Kommunikation eingesetzt, obwohl dies zu mehr Flexibilität und Modularität hätte führen können.
- **Kompatibilität**: Viele bestehende Applikationen und Bibliotheken wissen nichts von Netzwerken und sollen dennoch in einem verteilten Umfeld eingesetzt werden können. Dies wurde dadurch erreicht, dass nach einem erfolgten Verbindungsaufbau (der z.B. von einem anderen Prozess durchgeführt werden kann) Ein- und Ausgabe in gewohnter Weise (wie bei Dateien, Pipelines oder Terminal-Verbindungen) erfolgen können.

Die Semantik einer Kommunikation umschließt bei jeder Verbindung eine Teilmenge der folgenden Punkte:

1. Daten werden in der Reihenfolge empfangen, in der sie abgeschickt worden sind.
2. Daten kommen nicht doppelt an.
3. Daten werden zuverlässig übermittelt.
4. Einzelne Pakete kommen in der originalen Form an (d.h. sie werden weder zerstückelt noch mit anderen Paketen kombiniert).
5. Nachrichten außerhalb des normalen Kommunikationsstromes (*out-of-band messages*) werden unterstützt.
6. Die Kommunikation erfolgt verbindungs-orientiert, womit die Notwendigkeit entfällt, sich bei jedem Paket identifizieren zu müssen.

Die folgende Tabelle zeigt die Varianten, die von der Berkeley-Socket-Schnittstelle unterstützt werden:

Name	1	2	3	4	5	6
SOCK_STREAM	⋆	⋆	⋆		⋆	⋆
SOCK_DGRAM				⋆		
SOCK_SEQPACKET	⋆	⋆	⋆	⋆	⋆	⋆
SOCK_RDM	⋆	⋆	⋆	⋆		

(1: Reihenfolge korrekt; 2: nicht doppelt; 3: zuverlässige Übermittlung; 4: keine Stückelung; 5: *out-of-band*; 6: verbindungsorientiert.)

- *SOCK_STREAM* lässt sich ziemlich direkt auf TCP abbilden.
- *SOCK_STREAM* kommt den Pipelines am nächsten, wenn davon abgesehen wird, dass die Verbindungen bei Pipelines nur unidirektional sind.
- UDP wird ziemlich genau durch *SOCK_DGRAM* widergespiegelt.
- Die Varianten *SOCK_SEQPACKET* (TCP-basiert) und *SOCK_RDM* (UDP-basiert) fügen hier noch weitere Funktionalitäten hinzu. Allerdings fand *SOCK_RDM* nicht den Weg in den POSIX-Standard und wird auch von einigen Implementierungen nicht angeboten.
- Im weiteren Verlauf dieser Vorlesung werden wir uns nur mit *SOCK_STREAM*-Sockets beschäftigen.

```
int sfd = socket(domain, type, protocol);
```

- Bis zu einem gewissen Grad ist eine Betrachtung, die sich an unserem Telefonsystem orientiert, hilfreich.
- Bevor Sie Telefonanrufe entgegennehmen oder selbst anrufen können, benötigen Sie einen Telefonanschluss.
- Dieser Anschluss wird mit dem Systemaufruf *socket* erzeugt.
- Bei *domain* wird hier normalerweise *PF_INET* angegeben, um das IPv4-Protokoll auszuwählen. (Alternativ wäre etwa *PF_INET6* für IPv6 denkbar.)
- *PF* steht dabei für *protocol family*. Bei *type* kann eine der unterstützten Semantiken ausgewählt werden, also beispielsweise *SOCK_STREAM*.
- Der dritte Parameter *protocol* erlaubt in einigen Fällen eine weitere Selektion. Normalerweise wird hier schlicht 0 angegeben.

- Nachdem der Anschluss existiert, fehlt noch eine zugeordnete Telefonnummer. Um bei der Analogie zu bleiben, haben wir eine Vorwahl (IP-Adresse) und eine Durchwahl (Port-Nummer).
- Auf einem Rechner können mehrere IP-Adressen zur Verfügung stehen.
- Es ist dabei möglich, nur eine dieser IP-Adressen zu verwenden oder alle gleichzeitig, die zur Verfügung stehen.
- Bei den Port-Nummern ist eine automatische Zuteilung durch das Betriebssystem möglich.
- Alternativ ist es auch möglich, sich selbst eine Port-Nummer auszuwählen. Diese darf aber noch nicht vergeben sein und muss bei nicht-privilegierten Prozessen eine Nummer jenseits des Bereiches der wohldefinierten Port-Nummern sein, also typischerweise mindestens 1024 betragen.
- Die Verknüpfung eines Anschlusses mit einer vollständigen Adresse erfolgt mit dem Systemaufruf *bind*...

```
struct sockaddr_in address = {0};
address.sin_family = AF_INET;
address.sin_addr.s_addr = htonl(INADDR_ANY);
address.sin_port = htons(port);
bind(sfd, (struct sockaddr *) &address, sizeof address);
```

- Die Datenstruktur **struct** *sockaddr_in* repräsentiert Adressen für IPv4, die aus einer IP-Adresse und einer Port-Nummer bestehen.
- Das Feld *sin_family* legt den Adressraum fest. Hier gibt es passend zur Protokollfamilie *PF_INET* nur *AF_INET* (*AF* steht hier für *address family*).
- Bei dem Feld *sin_addr.s_addr* lässt sich die IP-Adresse angeben. Mit *INADDR_ANY* übernehmen wir alle IP-Adressen, die zum eigenen Rechner gehören.
- Das Feld *sin_port* spezifiziert die Port-Nummer.

```
struct sockaddr_in address = {0};
address.sin_family = AF_INET;
address.sin_addr.s_addr = htonl(INADDR_ANY);
address.sin_port = htons(port);
bind(sfd, (struct sockaddr *) &address, sizeof address);
```

- Da Netzwerkadressen grundsätzlich nicht von der Byte-Anordnung eines Rechners abhängen dürfen, wird mit *htonl* (*host to network long*) der 32-Bit-Wert der IP-Adresse in die standardisierte Form konvertiert. Analog konvertiert *htons*() (*host to network short*) den 16-Bit-Wert *port* in die standardisierte Byte-Reihenfolge.
- Wenn die Port-Nummer vom Betriebssystem zugeteilt werden soll, kann bei *sin_port* auch einfach 0 angegeben werden.

```
struct sockaddr_in address = {0};
address.sin_family = AF_INET;
address.sin_addr.s_addr = htonl(INADDR_ANY);
address.sin_port = htons(port);
bind(sfd, (struct sockaddr *) &address, sizeof address);
```

- Der Datentyp **struct** *sockaddr_in* ist eine spezielle Variante des Datentyps **struct** *sockaddr*. Letzterer sieht nur ein Feld *sin_family* vor und ein generelles Datenfeld *sa_data*, das umfangreich genug ist, um alle unterstützten Adressen unterzubringen.
- Bei *bind*() wird der von *socket*() erhaltene Deskriptor angegeben (hier *sfd*), ein Zeiger, der auf eine Adresse vom Typ **struct** *sockaddr* verweist, und die tatsächliche Länge der Adresse, die normalerweise kürzer ist als die des Typs **struct** *sockaddr*.
- Schön sind diese Konstruktionen nicht, aber C bietet eben keine objekt-orientierten Konzepte, wenngleich die Berkeley-Socket-Schnittstelle sehr wohl polymorph und damit objekt-orientiert ist.

```
listen(sfd, SOMAXCONN);
```

- Damit eingehende Verbindungen (oder Anrufe in unserer Telefon-Analogie) entgegengenommen werden können, muss *listen*() aufgerufen werden.
- Nach *listen*() kann der Anschluss „klingeln“, aber noch sind keine Vorbereitungen getroffen, das Klingeln zu hören oder den Hörer abzunehmen.
- Der zweite Parameter bei *listen*() gibt an, wieviele Kommunikationspartner es gleichzeitig klingeln lassen dürfen.
- *SOMAXCONN* ist hier das Maximum, das die jeweilige Implementierung erlaubt.

Anschlussdaten ermitteln

newsocket.c

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>

void print_ip_addr(in_addr_t ipaddr) {
   if (ipaddr == INADDR_ANY) {
      printf("INADDR_ANY");
   } else {
      uint32_t addr = ntohl(ipaddr);
      printf("%u.%u.%u.%u",
         addr>>24, (addr>>16)&0xff,
         (addr>>8)&0xff, addr&0xff);
   }
}

int main() {
   int sfd = socket(PF_INET, SOCK_STREAM, 0);
   if (sfd < 0) exit(1);
   if (listen(sfd, SOMAXCONN) < 0) exit(2);
   struct sockaddr address;
   socklen_t len = sizeof address;
   if (getsockname(sfd, &address, &len) < 0) exit(3);
   struct sockaddr_in * inaddr = (struct sockaddr_in *) &address;
   printf("This is the address of my new socket:\n");
   printf("IP address:  "); print_ip_addr(inaddr->sin_addr.s_addr);
   printf("\n");
   printf("Port number: %hu\n", ntohs(inaddr->sin_port));
}
```

```
struct sockaddr client_addr;
socklen_t client_addr_len = sizeof client_addr;
int fd = accept(sfd, &client_addr, &client_addr_len);
```

- Liegt noch kein Anruf vor, blockiert *accept*() bis zum nächsten Anruf.
- Wenn mit *accept*() ein Anruf eingeht, wird ein Dateideskriptor auf den bidirektionalen Verbindungskanal zurückgeliefert.
- Normalerweise speichert *accept*() die Adresse des Klienten beim angegebenen Zeiger ab. Wenn als Zeiger 0 angegeben wird, entfällt dies.

Ein einfacher Zeitdienst

timeserver.c

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>
#define PORT 11011
int main () {
   struct sockaddr_in address = {0};
   address.sin_family = AF_INET;
   address.sin_addr.s_addr = htonl(INADDR_ANY);
   address.sin_port = htons(PORT);
   int sfd = socket(PF_INET, SOCK_STREAM, 0);
   int optval = 1;
   if (sfd < 0 ||
        setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR,
                   &optval, sizeof optval) < 0 ||
        bind(sfd, (struct sockaddr *) &address,
             sizeof address) < 0 ||
        listen(sfd, SOMAXCONN) < 0) {
      perror("socket"); exit(1);
   }
   int fd;
   while ((fd = accept(sfd, 0, 0)) >= 0) {
      char timebuf[32]; time_t clock; time(&clock);
      ctime_r(&clock, timebuf);
      write(fd, timebuf, strlen(timebuf)); close(fd);
   }
}
```

timeserver.c

```
if (sfd < 0 ||
     setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR,
                &optval, sizeof optval) < 0 ||
     bind(sfd, (struct sockaddr *) &address,
          sizeof address) < 0 ||
     listen(sfd, SOMAXCONN) < 0) {
   perror("socket"); exit(1);
}
```

- Hier wird zusätzlich noch *setsockopt* aufgerufen, um die Option *SO_REUSEADDR* einzuschalten.
- Dies empfiehlt sich immer, wenn eine feste Port-Nummer verwendet wird.
- Fehlt diese Option, kann es passieren, dass bei einem Neustart des Dienstes die Port-Nummer nicht sofort wieder zur Verfügung steht, da noch alte Verbindungen nicht vollständig abgewickelt worden sind.

Ein Klient für den Zeitdienst

timeclient.c

```
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 11011
int main (int argc, char** argv) {
   char* cmdname = *argv++; --argc;
   if (argc != 1) {
      fprintf(stderr, "Usage: %s host\n", cmdname); exit(1);
   }
   char* hostname = *argv; struct hostent* hp;
   if ((hp = gethostbyname(hostname)) == 0) {
      fprintf(stderr, "unknown host: %s\n", hostname); exit(1);
   }
   char* hostaddr = hp->h_addr_list[0];
   struct sockaddr_in addr = {0}; addr.sin_family = AF_INET;
   memcpy((void *) &addr.sin_addr, (void *) hostaddr, hp->h_length);
   addr.sin_port = htons(PORT);
   int fd;
   if ((fd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
      perror("socket"); exit(1);
   }
   if (connect(fd, (struct sockaddr *) &addr, sizeof addr) < 0) {
      perror("connect"); exit(1);
   }
   char buffer[BUFSIZ]; ssize_t nbytes;
   while((nbytes = read(fd, buffer, sizeof buffer)) > 0 &&
         write(1, buffer, nbytes) == nbytes);
}
```

timeclient.c

```
char* hostname = *argv;
struct hostent* hp;
if ((hp = gethostbyname(hostname)) == 0) {
   fprintf(stderr, "unknown host: %s\n", hostname);
   exit(1);
}
char* hostaddr = hp->h_addr_list[0];
struct sockaddr_in addr = {0};
addr.sin_family = AF_INET;
memmove((void *) &addr.sin_addr, (void *) hostaddr, hp->h_length);
addr.sin_port = htons(PORT);
```

- Der Klient erhält über die Kommandozeile den Namen des Rechners, auf dem der Zeitdienst zur Verfügung steht.
- Für die Abbildung eines Rechnernamens in eine IP-Adresse wird die Funktion *gethostbyname*() benötigt, die im Erfolgsfalle eine oder mehrere IP-Adressen liefert, unter denen sich der Rechner erreichen lässt.
- Hier wird die erste IP-Adresse ausgewählt.