Internet Sockets

Socket-Programmierung mit C

Michael Dienert

Walther-Rathenau-Gewerbeschule
Freiburg

6. Juni 2018

Inhalt

HowTo und Compiler

- Wichtiges HowTo: Beej's Guide to Network Programming (Brian J. Hall, beej@beej.us)
- Compiler: gcc (GNU Compiler Collection)
- gcc compiliert C-Quelldatei (Endung .c) in ausführbare Binärdatei: **gcc hallo.c -o hallo**
- Name der Binärdatei: mit Option -o <name> setzen.
- Wird -o weggelassen, heisst die Binärdatei **a.out**

HowTo und Compiler

- Wichtiges HowTo: Beej's Guide to Network Programming (Brian J. Hall, beej@beej.us)
- Compiler: gcc (GNU Compiler Collection)
- gcc compiliert C-Quelldatei (Endung .c) in ausführbare Binärdatei: **gcc hallo.c -o hallo**
- Name der Binärdatei: mit Option -o <name> setzen.
- Wird -o weggelassen, heisst die Binärdatei **a.out**

HowTo und Compiler

- Wichtiges HowTo: Beej's Guide to Network Programming (Brian J. Hall, beej@beej.us)
- Compiler: gcc (GNU Compiler Collection)
- gcc compiliert C-Quelldatei (Endung .c) in ausführbare Binärdatei: **gcc hallo.c -o hallo**
- Name der Binärdatei: mit Option -o <name> setzen.
- Wird -o weggelassen, heisst die Binärdatei **a.out**

HowTo und Compiler

- Wichtiges HowTo: Beej's Guide to Network Programming (Brian J. Hall, beej@beej.us)
- Compiler: gcc (GNU Compiler Collection)
- gcc compiliert C-Quelldatei (Endung .c) in ausführbare Binärdatei: **gcc hallo.c -o hallo**
- Name der Binärdatei: mit Option -o <name> setzen.
- Wird -o weggelassen, heisst die Binärdatei **a.out**

HowTo und Compiler

- Wichtiges HowTo: Beej's Guide to Network Programming (Brian J. Hall, beej@beej.us)
- Compiler: gcc (GNU Compiler Collection)
- gcc compiliert C-Quelldatei (Endung .c) in ausführbare Binärdatei: **gcc hallo.c -o hallo**
- Name der Binärdatei: mit Option -o <name> setzen.
- Wird -o weggelassen, heisst die Binärdatei **a.out**

HowTo und Compiler

- Wichtiges HowTo: Beej's Guide to Network Programming (Brian J. Hall, beej@beej.us)
- Compiler: gcc (GNU Compiler Collection)
- gcc compiliert C-Quelldatei (Endung .c) in ausführbare Binärdatei: **gcc hallo.c -o hallo**
- Name der Binärdatei: mit Option -o <name> setzen.
- Wird -o weggelassen, heisst die Binärdatei **a.out**

Inhalt

Internet Sockets

- Die Layer4-Protokolle **TCP** und **UDP** sorgen für Datenaustausch zwischen entfernten Rechnern.
- Sie wurden so entwickelt, dass man mit Standard Ein- bzw. Ausgabeoperationen unter Unix Daten lesen- und schreiben kann.
- Ein Internet Socket ist je ein **Endpunkt** einer Verbindung zweier Rechner über das Internet.
- Der Internet Socket ist nur eine **logische** Struktur, d.h. er existiert nur in Software.
- Man kann sich Internet Sockets als **Dateischnittstellen** ins Internet vorstellen.

Internet Sockets

- Die Layer4-Protokolle **TCP** und **UDP** sorgen für Datenaustausch zwischen entfernten Rechnern.
- Sie wurden so entwickelt, dass man mit Standard Ein- bzw. Ausgabeoperationen unter Unix Daten lesen- und schreiben kann.
- Ein Internet Socket ist je ein **Endpunkt** einer Verbindung zweier Rechner über das Internet.
- Der Internet Socket ist nur eine **logische** Struktur, d.h. er existiert nur in Software.
- Man kann sich Internet Sockets als **Dateischnittstellen** ins Internet vorstellen.

Internet Sockets

- Die Layer4-Protokolle **TCP** und **UDP** sorgen für Datenaustausch zwischen entfernten Rechnern.
- Sie wurden so entwickelt, dass man mit Standard Ein- bzw. Ausgabeoperationen unter Unix Daten lesen- und schreiben kann.
- Ein Internet Socket ist je ein **Endpunkt** einer Verbindung zweier Rechner über das Internet.
- Der Internet Socket ist nur eine **logische** Struktur, d.h. er existiert nur in Software.
- Man kann sich Internet Sockets als **Dateischnittstellen** ins Internet vorstellen.

Internet Sockets

- Die Layer4-Protokolle **TCP** und **UDP** sorgen für Datenaustausch zwischen entfernten Rechnern.
- Sie wurden so entwickelt, dass man mit Standard Ein- bzw. Ausgabeoperationen unter Unix Daten lesen- und schreiben kann.
- Ein Internet Socket ist je ein **Endpunkt** einer Verbindung zweier Rechner über das Internet.
- Der Internet Socket ist nur eine **logische** Struktur, d.h. er existiert nur in Software.
- Man kann sich Internet Sockets als **Dateischnittstellen** ins Internet vorstellen.

Internet Sockets

- Die Layer4-Protokolle **TCP** und **UDP** sorgen für Datenaustausch zwischen entfernten Rechnern.
- Sie wurden so entwickelt, dass man mit Standard Ein- bzw. Ausgabeoperationen unter Unix Daten lesen- und schreiben kann.
- Ein Internet Socket ist je ein **Endpunkt** einer Verbindung zweier Rechner über das Internet.
- Der Internet Socket ist nur eine **logische** Struktur, d.h. er existiert nur in Software.
- Man kann sich Internet Sockets als **Dateischnittstellen** ins Internet vorstellen.

Internet Sockets

- Die Layer4-Protokolle **TCP** und **UDP** sorgen für Datenaustausch zwischen entfernten Rechnern.
- Sie wurden so entwickelt, dass man mit Standard Ein- bzw. Ausgabeoperationen unter Unix Daten lesen- und schreiben kann.
- Ein Internet Socket ist je ein **Endpunkt** einer Verbindung zweier Rechner über das Internet.
- Der Internet Socket ist nur eine **logische** Struktur, d.h. er existiert nur in Software.
- Man kann sich Internet Sockets als **Dateischnittstellen** ins Internet vorstellen.

Zwei Arten von Sockets: Stream- und Datagramm-Sockets

- **Stream Sockets** stellen eine sichere, 2-Wege-**Verbindung** für Datenströme zwischen 2 Rechnern her.
- das zugrunde liegende Protokoll ist **TCP**.
- die Verbindung zwischen den Hosts ist nicht physikalisch, d.h. sie basiert auf dem **paketorientierten** Protokoll **IP**.
- TCP **verbirgt** diese hässliche Paketorientierung.
- Datagram Sockets basieren auf **UDP**. Sie sind nicht verbindungsorientiert und erlauben das Versenden von bis zu 64kByte grossen Datenblöcken.
- nicht verbindungsorientiert = wegschicken und vergessen

Zwei Arten von Sockets: Stream- und Datagramm-Sockets

- **Stream Sockets** stellen eine sichere, 2-Wege-**Verbindung** für Datenströme zwischen 2 Rechnern her.
- das zugrunde liegende Protokoll ist **TCP**.
- die Verbindung zwischen den Hosts ist nicht physikalisch, d.h. sie basiert auf dem **paketorientierten** Protokoll **IP**.
- TCP **verbirgt** diese hässliche Paketorientierung.
- Datagram Sockets basieren auf **UDP**. Sie sind nicht verbindungsorientiert und erlauben das Versenden von bis zu 64kByte grossen Datenblöcken.
- nicht verbindungsorientiert = wegschicken und vergessen

Zwei Arten von Sockets: Stream- und Datagramm-Sockets

- **Stream Sockets** stellen eine sichere, 2-Wege-**Verbindung** für Datenströme zwischen 2 Rechnern her.
- das zugrunde liegende Protokoll ist **TCP**.
- die Verbindung zwischen den Hosts ist nicht physikalisch, d.h. sie basiert auf dem **paketorientierten** Protokoll **IP**.
- TCP **verbirgt** diese hässliche Paketorientierung.
- Datagram Sockets basieren auf **UDP**. Sie sind nicht verbindungsorientiert und erlauben das Versenden von bis zu 64kByte grossen Datenblöcken.
- nicht verbindungsorientiert = wegschicken und vergessen

Zwei Arten von Sockets: Stream- und Datagramm-Sockets

- **Stream Sockets** stellen eine sichere, 2-Wege-**Verbindung** für Datenströme zwischen 2 Rechnern her.
- das zugrunde liegende Protokoll ist **TCP**.
- die Verbindung zwischen den Hosts ist nicht physikalisch, d.h. sie basiert auf dem **paketorientierten** Protokoll **IP**.
- TCP **verbirgt** diese hässliche Paketorientierung.
- Datagram Sockets basieren auf **UDP**. Sie sind nicht verbindungsorientiert und erlauben das Versenden von bis zu 64kByte grossen Datenblöcken.
- nicht verbindungsorientiert = wegschicken und vergessen

Zwei Arten von Sockets: Stream- und Datagramm-Sockets

- **Stream Sockets** stellen eine sichere, 2-Wege-**Verbindung** für Datenströme zwischen 2 Rechnern her.
- das zugrunde liegende Protokoll ist **TCP**.
- die Verbindung zwischen den Hosts ist nicht physikalisch, d.h. sie basiert auf dem **paketorientierten** Protokoll **IP**.
- TCP **verbirgt** diese hässliche Paketorientierung.
- Datagram Sockets basieren auf **UDP**. Sie sind nicht verbindungsorientiert und erlauben das Versenden von bis zu 64kByte grossen Datenblöcken.
- nicht verbindungsorientiert = wegschicken und vergessen

Zwei Arten von Sockets: Stream- und Datagramm-Sockets

- **Stream Sockets** stellen eine sichere, 2-Wege-**Verbindung** für Datenströme zwischen 2 Rechnern her.
- das zugrunde liegende Protokoll ist **TCP**.
- die Verbindung zwischen den Hosts ist nicht physikalisch, d.h. sie basiert auf dem **paketorientierten** Protokoll **IP**.
- TCP **verbirgt** diese hässliche Paketorientierung.
- Datagram Sockets basieren auf **UDP**. Sie sind nicht verbindungsorientiert und erlauben das Versenden von bis zu 64kByte grossen Datenblöcken.
- nicht verbindungsorientiert = wegschicken und vergessen

Zwei Arten von Sockets: Stream- und Datagramm-Sockets

- **Stream Sockets** stellen eine sichere, 2-Wege-**Verbindung** für Datenströme zwischen 2 Rechnern her.
- das zugrunde liegende Protokoll ist **TCP**.
- die Verbindung zwischen den Hosts ist nicht physikalisch, d.h. sie basiert auf dem **paketorientierten** Protokoll **IP**.
- TCP **verbirgt** diese hässliche Paketorientierung.
- Datagram Sockets basieren auf **UDP**. Sie sind nicht verbindungsorientiert und erlauben das Versenden von bis zu 64kByte grossen Datenblöcken.
- nicht verbindungsorientiert = wegschicken und vergessen

File Descriptor

- Jede offene Dateischnittstelle auf einem Rechner hat einen sog. *file descriptor*, das ist ein integer-Wert.
- Der file descriptor hilft dem Kernel bei der Verwaltung offener Dateischnittstellen: d.h. Schnittstellen, in die man schreiben oder von denen man lesen kann
- vordefinierte File-Descriptoren sind: 0=stdin, 1=stdout, 2=stderr.
- öffent man einen Internet Socket (ab jetzt einfach nur Socket), erhält man vom Kernel einen file descriptor zurück.
- D.h. der Socket verhält sich wie eine offene Datei.

File Descriptor

- Jede offene Dateischnittstelle auf einem Rechner hat einen sog. *file descriptor*, das ist ein integer-Wert.
- Der file descriptor hilft dem Kernel bei der Verwaltung offener Dateischnittstellen: d.h. Schnittstellen, in die man schreiben oder von denen man lesen kann
- vordefinierte File-Descriptoren sind: 0=stdin, 1=stdout, 2=stderr.
- öffent man einen Internet Socket (ab jetzt einfach nur Socket), erhält man vom Kernel einen file descriptor zurück.
- D.h. der Socket verhält sich wie eine offene Datei.

File Descriptor

- Jede offene Dateischnittstelle auf einem Rechner hat einen sog. *file descriptor*, das ist ein integer-Wert.
- Der file descriptor hilft dem Kernel bei der Verwaltung offener Dateischnittstellen: d.h. Schnittstellen, in die man schreiben oder von denen man lesen kann
- vordefinierte File-Descriptoren sind: 0=stdin, 1=stdout, 2=stderr.
- öffent man einen Internet Socket (ab jetzt einfach nur Socket), erhält man vom Kernel einen file descriptor zurück.
- D.h. der Socket verhält sich wie eine offene Datei.

File Descriptor

- Jede offene Dateischnittstelle auf einem Rechner hat einen sog. *file descriptor*, das ist ein integer-Wert.
- Der file descriptor hilft dem Kernel bei der Verwaltung offener Dateischnittstellen: d.h. Schnittstellen, in die man schreiben oder von denen man lesen kann
- vordefinierte File-Descriptoren sind: 0=stdin, 1=stdout, 2=stderr.
- öffent man einen Internet Socket (ab jetzt einfach nur Socket), erhält man vom Kernel einen file descriptor zurück.
- D.h. der Socket verhält sich wie eine offene Datei.

File Descriptor

- Jede offene Dateischnittstelle auf einem Rechner hat einen sog. *file descriptor*, das ist ein integer-Wert.
- Der file descriptor hilft dem Kernel bei der Verwaltung offener Dateischnittstellen: d.h. Schnittstellen, in die man schreiben oder von denen man lesen kann
- vordefinierte File-Descriptoren sind: 0=stdin, 1=stdout, 2=stderr.
- öffent man einen Internet Socket (ab jetzt einfach nur Socket), erhält man vom Kernel einen file descriptor zurück.
- D.h. der Socket verhält sich wie eine offene Datei.

File Descriptor

- Jede offene Dateischnittstelle auf einem Rechner hat einen sog. *file descriptor*, das ist ein integer-Wert.
- Der file descriptor hilft dem Kernel bei der Verwaltung offener Dateischnittstellen: d.h. Schnittstellen, in die man schreiben oder von denen man lesen kann
- vordefinierte File-Descriptoren sind: 0=stdin, 1=stdout, 2=stderr.
- öffent man einen Internet Socket (ab jetzt einfach nur Socket), erhält man vom Kernel einen file descriptor zurück.
- D.h. der Socket verhält sich wie eine offene Datei.

IP-Adressen und Hostnamen

- Das Internet-Protokoll gibt es in 2 Versionen: IPv4 und IPv6
- Beispiel IPv4: 192.168.178.71/24
- Beispiel IPv6: 2003:e2:43ed:0:ca60:ff:fec7:b1f/64
- Ein Rechner kann mehrere Hardware-Netzwerkschnittstellen haben
- Eine Schnittstelle kann viele logische Adressen haben
- D.h.: ein Rechner hat u.U. ein Vielzahl an IP-Adressen
- Neben den Adressen können Rechner auch Namen haben, die über das **Domain Name System** in Adressen aufgelöst werden können

IP-Adressen und Hostnamen

- Das Internet-Protokoll gibt es in 2 Versionen: IPv4 und IPv6
- Beispiel IPv4: 192.168.178.71/24
- Beispiel IPv6: 2003:e2:43ed:0:ca60:ff:fec7:b1f/64
- Ein Rechner kann mehrere Hardware-Netzwerkschnittstellen haben
- Eine Schnittstelle kann viele logische Adressen haben
- D.h.: ein Rechner hat u.U. ein Vielzahl an IP-Adressen
- Neben den Adressen können Rechner auch Namen haben, die über das **Domain Name System** in Adressen aufgelöst werden können

IP-Adressen und Hostnamen

- Das Internet-Protokoll gibt es in 2 Versionen: IPv4 und IPv6
- Beispiel IPv4: 192.168.178.71/24
- Beispiel IPv6: 2003:e2:43ed:0:ca60:ff:fec7:b1f/64
- Ein Rechner kann mehrere Hardware-Netzwerkschnittstellen haben
- Eine Schnittstelle kann viele logische Adressen haben
- D.h.: ein Rechner hat u.U. ein Vielzahl an IP-Adressen
- Neben den Adressen können Rechner auch Namen haben, die über das **Domain Name System** in Adressen aufgelöst werden können

IP-Adressen und Hostnamen

- Das Internet-Protokoll gibt es in 2 Versionen: IPv4 und IPv6
- Beispiel IPv4: 192.168.178.71/24
- Beispiel IPv6: 2003:e2:43ed:0:ca60:ff:fec7:b1f/64
- Ein Rechner kann mehrere Hardware-Netzwerkschnittstellen haben
- Eine Schnittstelle kann viele logische Adressen haben
- D.h.: ein Rechner hat u.U. ein Vielzahl an IP-Adressen
- Neben den Adressen können Rechner auch Namen haben, die über das **Domain Name System** in Adressen aufgelöst werden können

IP-Adressen und Hostnamen

- Das Internet-Protokoll gibt es in 2 Versionen: IPv4 und IPv6
- Beispiel IPv4: 192.168.178.71/24
- Beispiel IPv6: 2003:e2:43ed:0:ca60:ff:fec7:b1f/64
- Ein Rechner kann mehrere Hardware-Netzwerkschnittstellen haben
- Eine Schnittstelle kann viele logische Adressen haben
- D.h.: ein Rechner hat u.U. ein Vielzahl an IP-Adressen
- Neben den Adressen können Rechner auch Namen haben, die über das **Domain Name System** in Adressen aufgelöst werden können

IP-Adressen und Hostnamen

- Das Internet-Protokoll gibt es in 2 Versionen: IPv4 und IPv6
- Beispiel IPv4: 192.168.178.71/24
- Beispiel IPv6: 2003:e2:43ed:0:ca60:ff:fec7:b1f/64
- Ein Rechner kann mehrere Hardware-Netzwerkschnittstellen haben
- Eine Schnittstelle kann viele logische Adressen haben
- D.h.: ein Rechner hat u.U. ein Vielzahl an IP-Adressen
- Neben den Adressen können Rechner auch Namen haben, die über das **Domain Name System** in Adressen aufgelöst werden können

IP-Adressen und Hostnamen

- Das Internet-Protokoll gibt es in 2 Versionen: IPv4 und IPv6
- Beispiel IPv4: 192.168.178.71/24
- Beispiel IPv6: 2003:e2:43ed:0:ca60:ff:fec7:b1f/64
- Ein Rechner kann mehrere Hardware-Netzwerkschnittstellen haben
- Eine Schnittstelle kann viele logische Adressen haben
- D.h.: ein Rechner hat u.U. ein Vielzahl an IP-Adressen
- Neben den Adressen können Rechner auch Namen haben, die über das **Domain Name System** in Adressen aufgelöst werden können

IP-Adressen und Hostnamen

- Das Internet-Protokoll gibt es in 2 Versionen: IPv4 und IPv6
- Beispiel IPv4: 192.168.178.71/24
- Beispiel IPv6: 2003:e2:43ed:0:ca60:ff:fec7:b1f/64
- Ein Rechner kann mehrere Hardware-Netzwerkschnittstellen haben
- Eine Schnittstelle kann viele logische Adressen haben
- D.h.: ein Rechner hat u.U. ein Vielzahl an IP-Adressen
- Neben den Adressen können Rechner auch Namen haben, die über das **Domain Name System** in Adressen aufgelöst werden können

IP-Adressen und Hostnamen

Beispiel:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state <schnipp/>
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc <schnipp/>
    link/ether c8:60:00:c7:0b:1f brd ff:ff:ff:ff:ff:ff
    inet 192.168.178.71/24 brd 192.168.178.255 scope global eth0
    valid_lft forever preferred_lft forever
    inet6 2003:e2:43ed:0:ca60:ff:fec7:b1f/64 scope global mngtmpaddr dynamic
    valid_lft 7136sec preferred_lft 1212sec
    inet6 fe80::ca60:ff:fec7:b1f/64 scope link
    valid_lft forever preferred_lft forever
3: eth1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc <schnipp/>
    link/ether 90:e2:ba:21:cc:8c brd ff:ff:ff:ff:ff:ff
    inet 10.10.0.1/24 scope global eth1
    valid_lft forever preferred_lft forever
    inet6 fe80::92e2:baff:fe21:cc8c/64 scope link
    valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc <schnipp/>
    link/ether 02:42:ae:78:64:95 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
    valid_lft forever preferred_lft forever
```

IP-Adressen und Hostnamen

- Alle Adressen (IPv4, IPv6, Port-Nummern) müsssen im Socket-Programm in speziellen **Datenstrukturen** gespeichert werden.
- Die Adressen werden in der DDN oder in der IPv6-Hexadezimalform eingegeben, werden intern aber binär dargestellt, müssen also **konvertiert** werden.
- Das Programm soll Rechner auch über ihren Namen erreichen können, d.h. das Programm soll Namen in Adressen auflösen können.
- Die ganze Arbeit mit der Adressaufbereitung erledigt die Bibliotheks-Funktion `getaddrinfo()`.
- `getaddrinfo()` liefert eine **verkettete Liste** mit allen möglichen Adressen des Hosts zurück.

IP-Adressen und Hostnamen

- Alle Adressen (IPv4, IPv6, Port-Nummern) müsssen im Socket-Programm in speziellen **Datenstrukturen** gespeichert werden.
- Die Adressen werden in der DDN oder in der IPv6-Hexadezimalform eingegeben, werden intern aber binär dargestellt, müssen also **konvertiert** werden.
- Das Programm soll Rechner auch über ihren Namen erreichen können, d.h. das Programm soll Namen in Adressen auflösen können.
- Die ganze Arbeit mit der Adressaufbereitung erledigt die Bibliotheks-Funktion `getaddrinfo()`.
- `getaddrinfo()` liefert eine **verkettete Liste** mit allen möglichen Adressen des Hosts zurück.

IP-Adressen und Hostnamen

- Alle Adressen (IPv4, IPv6, Port-Nummern) müsssen im Socket-Programm in speziellen **Datenstrukturen** gespeichert werden.
- Die Adressen werden in der DDN oder in der IPv6-Hexadezimalform eingegeben, werden intern aber binär dargestellt, müssen also **konvertiert** werden.
- Das Programm soll Rechner auch über ihren Namen erreichen können, d.h. das Programm soll Namen in Adressen auflösen können.
- Die ganze Arbeit mit der Adressaufbereitung erledigt die Bibliotheks-Funktion `getaddrinfo()`.
- `getaddrinfo()` liefert eine **verkettete Liste** mit allen möglichen Adressen des Hosts zurück.

IP-Adressen und Hostnamen

- Alle Adressen (IPv4, IPv6, Port-Nummern) müsssen im Socket-Programm in speziellen **Datenstrukturen** gespeichert werden.
- Die Adressen werden in der DDN oder in der IPv6-Hexadezimalform eingegeben, werden intern aber binär dargestellt, müssen also **konvertiert** werden.
- Das Programm soll Rechner auch über ihren Namen erreichen können, d.h. das Programm soll Namen in Adressen auflösen können.
- Die ganze Arbeit mit der Adressaufbereitung erledigt die Bibliotheks-Funktion `getaddrinfo()`.
- `getaddrinfo()` liefert eine **verkettete Liste** mit allen möglichen Adressen des Hosts zurück.

IP-Adressen und Hostnamen

- Alle Adressen (IPv4, IPv6, Port-Nummern) müsssen im Socket-Programm in speziellen **Datenstrukturen** gespeichert werden.
- Die Adressen werden in der DDN oder in der IPv6-Hexadezimalform eingegeben, werden intern aber binär dargestellt, müssen also **konvertiert** werden.
- Das Programm soll Rechner auch über ihren Namen erreichen können, d.h. das Programm soll Namen in Adressen auflösen können.
- Die ganze Arbeit mit der Adressaufbereitung erledigt die Bibliotheks-Funktion **`getaddrinfo()`**.
- `getaddrinfo()` liefert eine **verkettete Liste** mit allen möglichen Adressen des Hosts zurück.

IP-Adressen und Hostnamen

- Alle Adressen (IPv4, IPv6, Port-Nummern) müsssen im Socket-Programm in speziellen **Datenstrukturen** gespeichert werden.
- Die Adressen werden in der DDN oder in der IPv6-Hexadezimalform eingegeben, werden intern aber binär dargestellt, müssen also **konvertiert** werden.
- Das Programm soll Rechner auch über ihren Namen erreichen können, d.h. das Programm soll Namen in Adressen auflösen können.
- Die ganze Arbeit mit der Adressaufbereitung erledigt die Bibliotheks-Funktion `getaddrinfo()`.
- `getaddrinfo()` liefert eine **verkettete Liste** mit allen möglichen Adressen des Hosts zurück.