Netzwerk-Programmierung

Programmieren mit sockets

Jan Krüger
jkrueger@cebitec.uni-bielefeld.de
Alexander Sczyrba
asczyrba@cebitec.uni-bielefeld.de

Übersicht

- Datentypen und Konvertierung
- Minimaler Client
- Minimaler Server

Berkeley sockets

- sockets API, erstmals in 4.2BSD (1983)
- Einfache Implementierung:
  - TCP/IP
  - named pipes
  - OSI protocols
  - . . .
- C Funktionen
- Information über Adressen und Ports in structs
- Datentypenkonversion mit speziellen Funktionen

Namen und Adressen

- Netzwerkinterface durch IP-Adresse eindeutig identifiziert
- dotted quad-Notation: 129.70.132.229
- “lesbare“ Namen ermittelt durch Domain Name System (DNS)

  www.Uni-Bielefeld.DE → 129.70.240.4
- Keine Bijektion!
  - 173.194.69.147 → bk-in-f147.1e100.net
- Keine mathematische Funktion!
- www.google.com → (173.194.69.106, 173.194.69.147 . . .)

Adresskonvertierung

- `import socket`
- Binär-Representation für socket-Funktionen:

```
iaddr = socket.inet_aton('129.70.240.4');
```

- Vice versa:

```
dotquad = socket.inet_ntoa(iaddr);
```

- Alternative:

```
dotquad = socket.gethostbyname('www.uni-
bielefeld.de');
```

- Hostname and Aliase ermitteln:

```
(name, aliaslist, addresslist) =
socket.gethostbyaddr(iaddr|HOSTNAME, AF_INET);
```

Aufgaben

- Transformiert die folgenden Namen in Binär-Repräsentation und wieder zurück in dotted quad-Notation. Löst den Namen und die Aliase ausgehend von der dotted quad-Notation auf. Führt das Skript mehrere Male aus.
  Was sind die Unterschiede?

```
hosts = ['goldfinger',
         'goldfinger.techfak.uni-bielefeld.de',
         'www.ebay.com',
         'www.uni-paderborn.de',
         'www.cnn.com',
         'www.bielefeld.de']
```

- Vergleicht eure Ergebnisse mit denen vom Programm `dig`:

```
> dig www.ebay.com
```

Weitere Funktionen

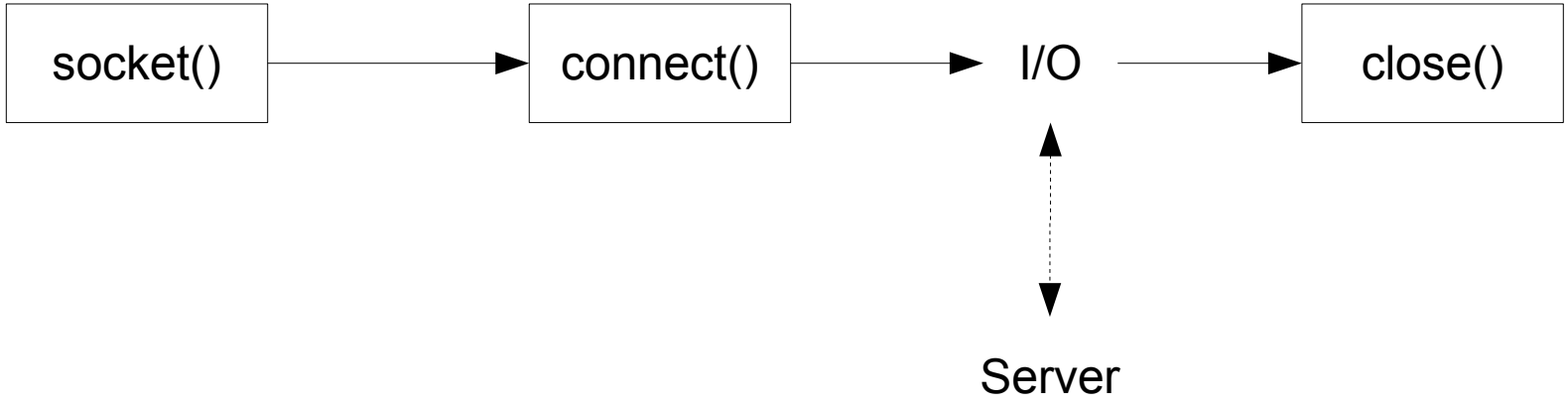
- Protokolle (vgl. `/etc/protocols`)

```
proto = socket.getprotobyname('tcp');
```

- Services (vgl. `/etc/services`)

```
service = socket.getservbyname('daytime'[, 'tcp']);
daytime = socket.getservbyport(13[, 'tcp']);
```

Arbeitsweise Client



Client-Code

- Erzeugen eines sockets:
  ```
  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM,
  socket.IPPROTO_TCP)
  ```
- socket-Adresse: Kombination aus Adresse und Port als 2-Tupel
  ```
  sockaddr = (dottquad|hostname, port);
  ```
- Verbindung herstellen:
  ```
  s.connect((<HOST>,<PORT>))
  ```
- Socket schließen:
  ```
  s.close()
  ```

Daten senden und lesen (Python3)

- Daten lesen:
  `bytes = s.recv(<BufferSize>)`
- Daten senden: →
  `sentbytes = s.send(bytes)`
- Keine Gewährleistung, dass alles gesendet/gelesen wurde!

`bytes()`

- `b“textTEXTtext“`
- `bytes → str:`
  `bytes.decode(<ENCODING>)`
  `bytes.decode('utf-8')`
- `str → bytes:`
  `bytes(<STR>,<ENCODING>)`
  `bytes(msg, 'utf-8')`

Aufgaben

- Mach Dich mit dem ServiceServer (Material zu dieser Übung) vertraut und teste beide Services mit `telnet`
  - Welche Services bietet der Server?
- Schreibe ein Client-Programm, das eine Verbindung aufbaut, ein Kommando absetzt (optional), alle Daten liest und die Verbindung wieder beendet. Der Zielrechner, der Zielport und das abzusetzende Kommando sollen als Argumente übergeben werden können.
  - Probiere den Client mit dem ServiceServer aus.
  - Teste den Client auch mit dem WebServer der Technischen Fakultät. Welcher Port, welches Protokoll und was für ein Kommando muss benutzt werden, um sich die Einstiegsseite anzeigen zu lassen?

Socket- Informationen

- Port wird dynamisch zugewiesen (*ephemeral port*)
- Beliebiges Interface bei *multihomed host*
- Informationen über sockets ermitteln:

```
mysockinfo = s.getsockname()
hissockinfo = s.getpeername()
```

- Rückgarewert bei IP-Sockets: 2-Tupel

```
(hostaddr, port)
```

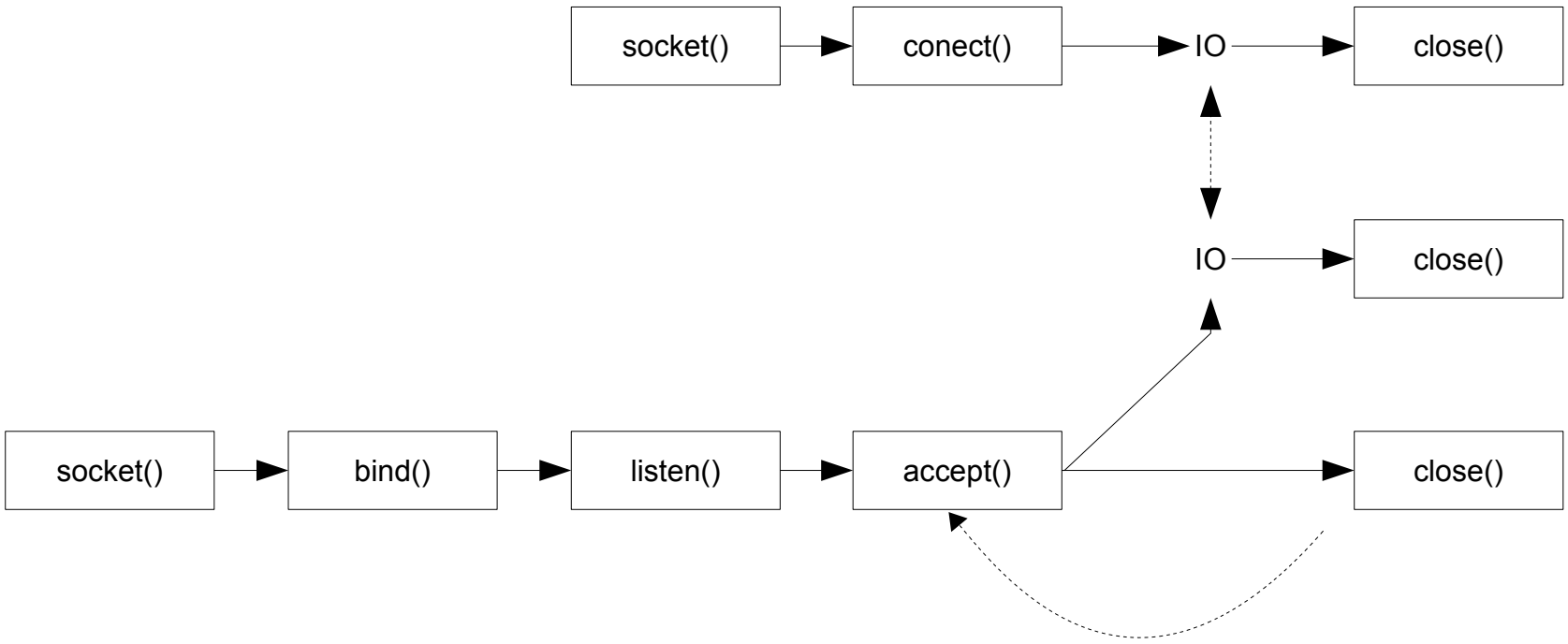
Aufgaben

- Erweitere das Programm aus der letzten Aufgabe so, daß alle Daten zur Verbindung angezeigt werden. Zur Erinnerung:

  socket pair: (IP-AdresseL, PortL, IP-AdresseR, PortR)

- Rufe das Programm mehrfach auf. Was ist zu beobachten?

Arbeitsweise Server



Server-Code, Teil 1

- `s = socket.socket(...)` wie im Client
- Socket an Port/Adresse binden:
  `socket.bind((<host>, <port>))`
- Passive open und backlog:
  `socket.listen(socket.SOMAXCONN)`
- Tatsächliche Größe des *backlog*s abhängig vom Betriebssystem

Server-Code, Teil 2

- Verbindungen entgegennehmen:

  ```
  (inSocket, conInfo) = s.accept()
  ```

- `accept()` blockiert, bis Verbindung hergestellt
- `conInfo` enthält Informationen über peer
- Typischerweise in `try-finally`-Block mit Schleife
- `inSocket` zum lesen/schreiben verwenden
- Am Ende Verbindungs-Socket schließen:

  ```
  inSocket.close()
  ```

Server-Code, cont.

- typischer Server-Code:

```
s = socket.socket(…)
s.bind(…)
s.listen(…)

try:
    while 1:
        inSocket, addr = s.accept()
        #CODEcodeCODEcodeCODE
finally:
    inSocket.close()
```

Aufgaben

- Schreibe einen Server, der auf Verbindungen wartet, zwei Zeilen Text sendet und dann die Verbindung schließt. Die erste Zeile soll den Client begrüßen, die zweite soll die aktuelle Uhrzeit ausgeben:

```
hello goldfinger.TechFak.Uni-Bielefeld.DE, nice to meet you
it's Mon Jun 2 15:14:17 2003
```

- Du kannst den Server entweder mit dem Client aus der letzten Aufgabe oder mit dem Programm `telnet` testen.