Mini-Kursus Netzwerkprogrammierung

Lidia Khmylko, Wolfgang Menzel

21.2.2011

Der Internet-Standard: TCP/IP

- Der größte Teil des Internet-Verkehrs verwendet das *transfer control protocol/internet protocol*.
- Alle wichtigen Protokolle setzen auf TCP auf (HTTP, FTP, SMTP, ...)
- Als Verbindung zwischen vernetzten Rechnern dienen *Sockets*.
- Diese können von den meisten Programmiersprachen aus benutzt werden (insbesondere in Prolog).
- Der Anwendungen: IRC, Maildienste, Nachrichten, remote administration, file sharing, Industriespionage ...

Sockets

- Socket = (Rechnername:Portnummer)
- z.B.: www.heise.de:80, localhost:4040, 196.32.4.124:51234
- Client = ein Programm, das einen bestimmten Socket anspricht
- z.B.: Browser, mail client, filesharing client…
- Server = ein Programm, das solche Verbindungen annimmt
- z.B.: Webserver, Mailserver, FTP-Server

•

Ports

Die Nummer eines Ports gibt Hinweise, welches Protokoll dort abläuft:

- *reserved ports:* 0–1023
- *registered ports:* 1024–49151
- *private ports:* 49152–65535

(Alle registrierten Ports sind aufgezählt in:
`http://www.iana.org/assignments/port-numbers`)

Socket-Kommunikation in Prolog

Ein Server muß den TCP-Zustand *listen* einnehmen:

- Socket-Objekt instanziieren:
  ```
  new(@socket, socket(4444)).
  ```
- Festlegen, was mit eingehenden Daten passiert:
  ```
  send(@socket, input_message,
       message(@prolog, write, @arg1)).
  ```
- Am Port horchen:
  ```
  send(@socket, listen).
  ```

Socket-Kommunikation in Prolog

Ein Klient muß an der richtigen Adresse *anklopfen*:

- Socket-Objekt instanziieren
  ```
  new(@socket, socket(tuple(Server,4444))).
  ```
- Verbindung herstellen:
  ```
  send(@socket, connect).
  ```
- Daten senden:
  ```
  send(@socket, format, 'GET index.html').
  ```
- Verbindung schließen:
  ```
  send(@socket, close).
  ```

Mini-Demonstration von Sockets

- In `/informatik/home/khmylko/playground`
- …starten Sie `./client` und `./server` in zwei Shells.
- Mit dem Prädikat `chat/1` können Daten
  vom einen Prolog-Prozeß zum anderen gesandt werden:
  `chat('Hi! Do you come here often?').`
- (Aber nicht vom anderen zum einen!)

I want my money back!

- Die client-server-Demo hat einen schweren Fehler:
  es gibt nur einseitige Kommunikation.
- Schreiben Sie auch in server.pl ein Prädikat chat/1, das antworten soll.
- Was passiert, wenn man dieses aufruft?
- Offenbar behandelt Prolog die Sockets nicht symmetrisch.
- Was ist der Unterschied zwischen eingehenden
  und ausgehenden Nachrichten?

Des Rätsels Lösung

- socket-Objekte sind viel allgemeiner verwendbar als nur für one-on-one-Kommunikation.
- Der horchende Socket horcht auch nach dem `connect` weiter und kann weitere Verbindungen eingehen.
- Um mit mehreren Partnern gleichzeitig reden zu können, muß man mehrere ausgehende Verbindungen unterscheiden.
- Jede dieser Verbindungen wird durch ein neues Objekt der Klasse socket repräsentiert.
- Wie kommen wir an das betreffende Objekt heran?

Multiplexing mit Sockets

- Die Klasse `socket` versteht eine weitere Botschaft: `accept_message`.
- Wann wird die ‘accept message’ ausgeführt? Schlagen Sie es nach!
- Fügen Sie ein:

```
send(@s, accept_message,
     message(@prolog, accept, @receiver, @arg1))
```

- Schreiben Sie außerdem ein Prädikat `accept/2`...
- ...und finden Sie heraus, welche Argumente es übergeben bekommt.

Horchen und Antworten gleichzeitig

- Indem wir uns den Antwortsocket merken, können wir endlich antworten.
- Zum Beispiel so:

```
accept(S1,S2):-
  assert(incoming(S1)),
  assert(outgoing(S2)).

chat(X):-
  outgoing(Out),
  atom_concat(X,'\n',Line),
  send(Out, format, Line).
```