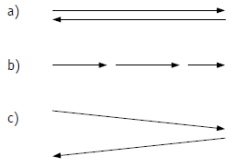
Socketkommunikation

Kommunikation

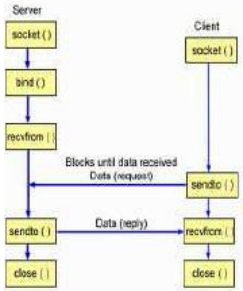
Paradigmen

1. Verbindungsorientierter Dienst eines (symmetrischen) Protokolls --> Sockets oder TCP
2. Message Passing --> von UDP bis e-Mail
3. Procedure Call --> in allen "remote" Varianten

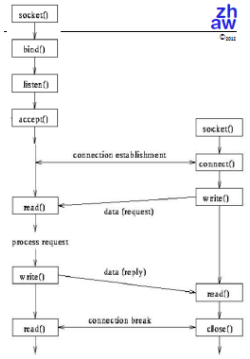


Socket

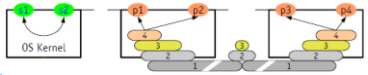
- Original-Unix-Feature
- Konzept bei fast allen anderen Betriebssystemen auch realisiert
- Üblicherweise offeriert über Kernel-API
- Systemweit adressierbar
- Verbindungslose Kommunikation



- Verbindungsorientierte Kommunikation

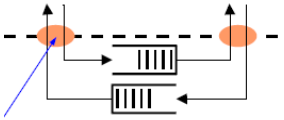


- Der klassische socket mausert sich zum TCP/UDP Port
- Port ist systemweit adressierbar: <ip>:<port>
- Funktionalität am API entspricht weitgehend dem bisherigen socket
- Neu sind Ausnahmesituationen wegen möglicher Netzwerkprobleme



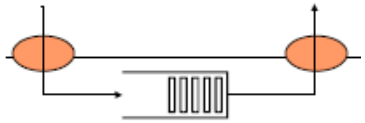
Dienst eines symmetrischen Protokolls

- Beispiele: Klassisches socket / Transportdienst (Schicht 4, TCP)
- Dienst ist charakterisiert durch Modell "Bidirectional Queue"
- Zwei unabhängige, asynchron arbeitende Übertragungskanäle
- An jeder Dienstschnittstelle zwei asynchron voneinander funktionierende Service-Primitiven

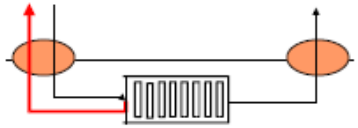


- Zuverlässigkeit: kein Verlust; Reihenfolge OK
- Leistung, Effizienz: "Pipelining" (Sliding Window Protocols)

Flusskontrolle



- Offene Warteschlange: lässt sich **unendlich** füllen
- **Keine** Flusskontrolle!



- **Endliche** Warteschlange
- Gibt "Back Pressure", wenn voll (Service-Primitive blockiert)

- Sliding Window Protokolle verwalten die Warteschlangenkapazität
- Abnahme-Engpass an der Empfänger-Dienstschnittstelle bewirkt Auffüllen der Warteschlange, dies wiederum bewirkt "Back Pressure" an der Sender-Dienstschnittstelle

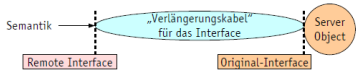
Verallgemeinerung des Modells

- Bounded Buffer, Reader / Writer

Write blocks if...		Read blocks if ...	Interaktion
Queue Full		Queue Empty	Asynchron
Buffer Full		Buffer Empty	Asynchron
Always, except Reader Waiting		Always, except Writer Waiting	Synchron

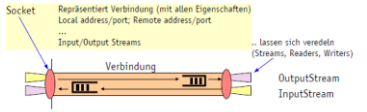
- **Synchrone Interaktion:**
  - Programmierung (Methodenaufruf)
  - Call/Accept-Mechanismus (ADA)
  - CSP, Prozess-Algebra
- **Asynchrone Interaktion:**
  - Dienste über Netzwerke (Datagramm, Mail)
  - Also auch RPC/RMI??

Remote Object



- Was muss das "Verlängerungskabel" können, damit irgendeine Original-Semantik genau so auch beim Remote Interface zu haben ist?
  - Das "absolut transparente Verlängerungskabel" ist eine Illusion
- Welche (Original-) Semantik lässt sich am einfachsten über einen gängigen Transportdienst "verlängern"
  - Ein Request-Response-Protokoll, also der "Procedure Call"
- **TCP: beliebte Basis für verschiedenste Protokolle zwischen verteilten Objekten**

TCP in der Java-Programmierung




```
Socket c = new Socket(String host, int port);
```

... macht aktiv Verbindungsaufbau

```
ServerSocket p = new ServerSocket(int port);
Socket s = p.accept();
```

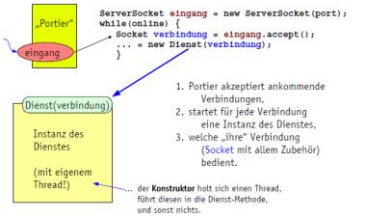
... macht passiv Verbindungsaufbau

```
s.close();
```

Umgang mit Sockets

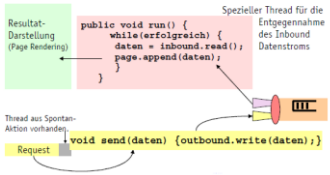
1. Socket-Konstruktor versucht Verbindungsaufnahme throws UnknownHostException, IOException, sollte ihm dies nicht gelingen
2. socket.getInputStream() bzw. socket.getOutputStream() liefern Anschlusspunkte für den Datenaustausch, die nach passender Veredelung sofort betriebsbereit sind
3. istream.close() bzw. ostream.close() ostream.flush() schliesst die Streams (leer die Pufferspeicher)
4. socket.close() beendet die TCP-Verbindung (Verbindungsaufbau)

TCP auf der Server-Seite



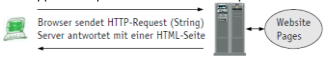
TCP auf der Client-Seite

„Input" und „output" -Streams rauschen i.a. unabhängig voneinander.



TCP mit HTTP

- Das Applikationsprotokoll für WWW ist http



- Client ist zumeist ein Browser, kann aber ebensogut irgendein Programm sein
- Die Interpretation von HTTP-Requests kann vom Web-Server an Servlets delegiert werden

Elemente von http

(-protocol headers)

Http-Methoden, Argumente

- GET actual „retrieve" operation. Conditionally returns response to client to obtain copy from cache
- POST send data to an URL, requesting it to do something with it (forms/CGI operations)
- HEAD get document info for cache freshness test
- PUT upload operation
- DELETE erase a resource behind an URL
- TRACE for debugging purposes
- CONNECT relay to another host (SSL through Proxy)

Wir werden darauf zurückkommen bei REST.

- seltener: CHECKOUT/CHECKIN; LINK/UNLINK; TEXTSEARCH; SHOWMETHOD; SPACEJUMP

```
GET /path/to/cgifile?a1=v1&a2=dies+und+das+%E4%F6%FC
Accept: image/gif, image/jpeg, text/*
```

GET: Argument passed on call command string
POST: Argument fed through stdin (forms data)

Answer: produced on stdout

TCP mit HTTPS (SSL)

- HTTPS ist genau wie HTTP, ausser dass die TCP-Sockets mit TLS (Transport Layer Security), alias **SSL**, ausgerüstet sind

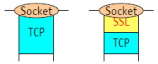
- Übertragungskanal ist kryptographisch gesichert
- Server "beweist" seine Identität
- Server kann vom Client dessen Authentifizierung verlangen

Interludium: X.509 Public Key System

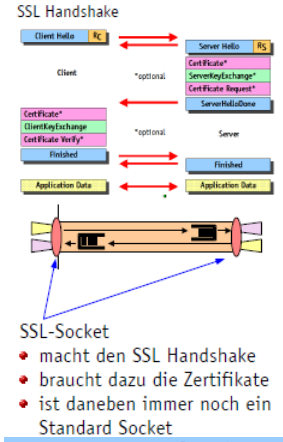
TODO: Seite 18

SSL-Ausrüstung für TCP in Java

- SSL im Schichtenmodell
- Socket ist Dienstschnittstelle von TCP

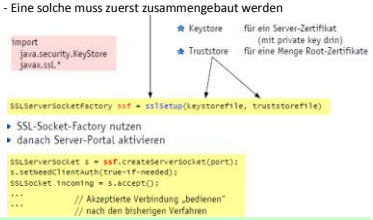


- SSL hast sich dazwischengeklemmt, jedoch: TCP bleibt TCP (das Protokoll)
- Der Socket ist immer noch die Dienstschnittstelle von TCP

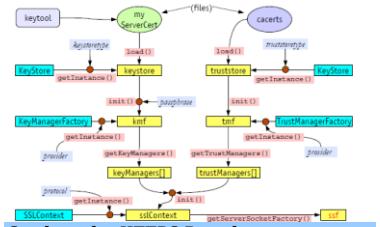


Umgang mit SSL-Sockets (serverseitig)

- Ein SSL-Socket ist mit Hilfe einer SSL-Socket-Factory zu erhalten
- Eine solche muss zuerst zusammengebaut werden

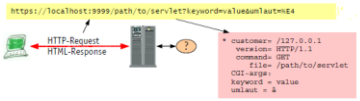


Ablauf / Organisation sslSetup



Struktur des HTTPS-Portals

Serverseitige Möglichkeiten



- Die Interpretation des HTTP-Requests lässt alles zu, was ein "normaler" Web-(Applikations)-Server kann:
  - File as-is zurückgeben
  - CGI-Programm in command-shell starten
  - Servlet-Code ausführen
    - in gemeinsamem Servlet für alle Requests
    - in separatem Servlet pro Request

Web-Applikations-Server

- Es existieren einige Web-Applikations-Server, welche die Servlet/JSP-Spezifikation (zumindest Versionen 3.0/2.5/2.1) erfüllen, darunter ist auch Apache Tomcat
- --> Servlet Container
- Servlet leben in einer "Sandbox", sind also in Sachen Sicherheit einigermassen unter Kontrolle
- Komplexere Web-Applikationen bestehen aus einer Menge Servlets und JSPs und brauchen etwas mehr Organisation
- JSPs sind Servlets, mit einem kleinen Unterschied:
  - Servlet = Java-Programmtext mit HTML als String-Literalen
  - JSP = HTML mit Markup-Tags, welche Java-Code enthalten
- JSPs lassen sich einfach im Container deponieren, der Server übernimmt deren Compilation automatisch

Weitere Details

HTTP-Authentifizierung

- Ein Servlet-Context kann so konfiguriert werden, dass der Client Userid/Passwort eingeben muss, um bestimmten Zugriff zu erhalten
- Dazu sind verschiedenste Benützer-verzeichnisse einsetzbar (XML-File, RDBMS, LDAP, etc.)
- Neben dem Standard Passwort-Dialog (des Browsers) kann auch ein spezifisch gestalteter Dialog (HTML-Formular) eingesetzt werden

SSL-Authentifizierung

- Für HTTPS wird z.B. in Tomcat ein "Connector" konfiguriert
- Client-seitige Authentifizierung ist an die SSL-Connection gebunden, kann also nicht vom Servlet-Context "mittendrin" verlangt werden
- Angaben aus dem Client-Zertifikat lassen sich dem Request entlocken

- ein http-POST request könnte an die doGet-Methode delegiert werden
- bei http-POST kommen die Daten als Stream

```
BufferedReader in = req.getReader();
String contentType = req.getContentType();
int contentLength = req.getContentLength();
```

MIME-Type

(so lässt sich z.B. ein „Upload" realisieren, anstelle von http-PUT)

- Ferner Funktionieren:

Server-Side Includes: Eine generierte Response wird auf weitere Servlet Calls untersucht

Chaining: Ein Servlet kann seine begonnene Arbeit an ein weiteres Servlet weitergeben (z.B. zwecks Filterung, Personalisierung)

Session Management: Session Tracking über mehrere Servlets hinweg ist gut unterstützt. Dadurch kann allenfalls auf Cookies für diesen Zweck verzichtet werden

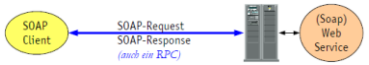
Servlet- und Request- Life Cycle

- Servlet Container instantiiert das Servlet-Objekt und ruft dessen Methode init() auf
- Da kann auf aktuelle Konfigurationen zugegriffen werden: ServletConfig und ServletContext
- Jeder Client-Request führt seinen Thread durch die entsprechende Service-Methode doGet(), doPost(), doHead(), etc. des genau einmal vorhandenen Servlet-Objekts
- Irgendwann befiehlt der Servlet Container destroy()

- http-GET ist "idempotent": wird normalerweise für "Retrieval" verwendet (allenfalls parameterisiert)
- http-POST ist nicht idempotent: Mitgelieferte Daten und Parameter bewirken eine Transaktion mit Effekt

WebService mit SOAP

- Client ist nicht notwendigerweise ein Browser, auch wenn er ein GUI aufweist (z.B. "Rich Client")
- HTML als Response-Format nicht unbedingt das passende
- HTTP transportiert genauso gerne XML, sowohl beim Request als auch bei der Response



- Jetzt haben wir schon wieder einen RPC, das "Simple Object Access Protocol"

	Anwendung
6	SOAP/XML
5	HTTP
4	TCP(+SSL)

- XML macht SOAP plattformunabhängig

Zusammenfassung

- In verteilten Anwendungssystemen wird zwischen Client- und Server-Funktionen vornehmlich über irgendeine Variante von "Remote Procedure Call" kommuniziert
- Die gemeinsame Basis ist weitgehend TCP, was neben RPC auch andere Protokolle zulässt (z.B. Streaming)
- TCP ist dermassen zentral, dass Sicherheitsanforderungen an Client/Server-Verbindung auf dieser Stufe erfüllt werden (SSL)
- Aus der Entwicklung des WWW ergibt sich das ubiquitäre Protokoll "http". Aus betrieblichen Gründen ist es bald einmal die einzige Client-Server-Verbindung, welche von Firewalls noch toleriert werden
- "http-Server" bieten neben klassischer WWW-Funktionalität auch umfassende Dienste für Server-seitige Programmierung: Servlet Container. In diesem Sinne steckt darin eine Menge Middleware "http" wird schliesslich Träger für den "ultimativen" RPC, nämlich SOAP
- SOAP erhält Konkurrenz durch REST (Representational State Transfer"