

## Vor zwei Wochen

Themen der letzten Vorlesung:

- Classless InterDomain Routing (CIDR)
- IPv6
- TCP und UDP
- Socket-Primitiva
- Einfacher Client/Server mit Sockets
- Hyper Text Transfer Protocol (HTTP)

## Wiederholung: URLs

Eine URL beschreibt den Pfad zu einer Ressource:

```
http : // www.beispiel.de /jokes/random ? cat=good%20ones
```

```
scheme  T T  authority          path-abempty  T  query
```

```
Protokoll      Server          Pfad/Programm      Argumente
```

## Wiederholung: HTTP-Nachrichten

- Zwei Arten von Nachrichten: Anfragen und Antworten
- Beide Nachrichten haben dieselbe Struktur

Schema:

```
<Request-Line>
<Feld-Name>: <Feld-Wert>
...
[ <Rumpf> ]
```

Die **Request-Line** unterscheidet Anfragen und Antworten.

## Wiederholung: Anfrage und Antwort

Die **Request-Line** für Anfragen:

```
Request-Line = Method " " Request-URI " " HTTP-Version CRLF
```

**Method** beschreibt Operation, die auf der Ressource **Request-URI** durchgeführt wird.

*Beispiel:* **GET** sendet den Inhalt des unter der gegebenen URI zu findenden Dokumentes

## Die POST-Methode

**POST** wird verwendet, um Daten zum Server zu übertragen.

- Daten sind Eingabe für an der URI hinterlegte Funktion
- Beispiel: Daten aus HTML-Formular können per **POST** übertragen werden
- Daten werden im **message-body** übertragen

## Die PUT- und DELETE-Methode

**PUT**: Daten an Server übertragen und unter gegebener URI ablegen

- Beispiel: Dokumente auf einen Server hochladen

**DELETE**: Daten unter gegebener URI vom Server löschen

- Beispiel: Dokumente auf dem Server löschen

## HTTP-Antworten

Die Antwort wird durch eine **status-Line** eingeleitet:

**status-Line** = **HTTP-Version** " " **Status-Code** " " **Reason-Phrase** CRLF

- Fehlercode (dreistellig) in **status-Code**
- **Reason-Phrase** Beschreibung des Fehlers

## Wichtige Status-Codes

<b>Code</b>	<b>Bedeutung</b>
200	Die Anfrage wurde erfolgreich bearbeitet
202	Anfrage wurde angenommen; Bearbeitung dauert an
301	Angeforderte URI ist unter neuer dauerhafter URI erreichbar
400	Syntax-Fehler in der Anfrage
403	Authorisierung erforderlich; Zugriff verweigert
404	Angeforderte Ressource nicht gefunden
500	Interner Serverfehler

## Beispiel zu GET

```
[0 knaue1@albert ~] telnet www-pu.informatik.uni-tuebingen.de 80
Trying 134.2.12.37...
Connected to www-pu.informatik.uni-tuebingen.de.
Escape character is '^]'.
GET /abc.html HTTP/1.1
Host: www-pu.informatik.uni-tuebingen.de

HTTP/1.1 200 OK
Date: Mon, 24 Oct 2005 09:02:35 GMT
Server: Apache/1.3.33 (Unix) PHP/4.3.11 mod_ssl/2.8.23
Last-Modified: Tue, 19 Jul 2005 16:35:51 GMT
Accept-Ranges: bytes
Content-Length: 7627
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
...
```

## Beispiel zu POST

```
[0 knaue1@albert ~] telnet www-pu.informatik.uni-tuebingen.de 80
Trying 134.2.12.37...
Connected to www-pu.informatik.uni-tuebingen.de.
Escape character is '^]'.
POST /cgi-bin/all-env.scm HTTP/1.1
Host: www-pu.informatik.uni-tuebingen.de
Content-Length: 10

Hallo=Welt

HTTP/1.1 200 OK
Date: Mon, 24 Oct 2005 09:17:21 GMT
Server: Apache/1.3.33 (Unix) PHP/4.3.11 mod_ssl/2.8.23
Transfer-Encoding: chunked
Content-Type: text/html

c5b
<html><body>
...
0
```

## Persistente Verbindungen

Bisher: Für jede Anfrage eine TCP-Verbindung

⇒ großer Overhead

Persistente Verbindungen:

- Feature von HTTP 1.1
- TCP-Verbindung für weitere Anfragen verwenden
- Header zeigt an, daß die Verbindung geschlossen wird:  
**Connection: close**

Außerdem *Pipelining*:

- Mehrere Anfragen direkt hintereinander
- Antworten treffen in Reihenfolge der Anfragen ein

## Serverseitige Webprogrammierung

## Serverseitige Webprogrammierung

Bisher: Ressourcen auf dem Webserver sind statisch

Gewünscht: *dynamischer Inhalt* und echte Applikationen

- Ressource/Webseite wird extra für die Anfrage erstellt
- Eingaben des Benutzers entgegennehmen
- Erstellung der Webseite durch ein Programm

⇒ Brauchen Schnittstelle zwischen Webserver und dem Programm zur Erstellung der Seite

## Techniken

Realisierung einer Webapplikation ist möglich als

- Externes und eigenständiges Programm
- Zuladbare interne Funktion des Webservers
- Programm in einer *Domain Specific Language (DSL)*

Wir beschäftigen uns mit:

- Common Gateway Interface (CGI)
- PHP: Hypertext Preprocessor (PHP)
- continuation-basierter Webprogrammierung

## Common Gateway Interface

CGI beschreibt Kommunikation zwischen Webserver und externem Programm.

- Sprachunabhängig
- Kommunikation über Standard-Ein/Ausgabe
- Kommunikation über Umgebungsvariablen
- Wird praktisch von allen Webservern unterstützt

## CGI-Programme

In der Praxis:

- CGI-Programme liegen in einem Extraverzeichnis
- URLs, die in dieses Verzeichnis zeigen, identifizieren CGI-Programme

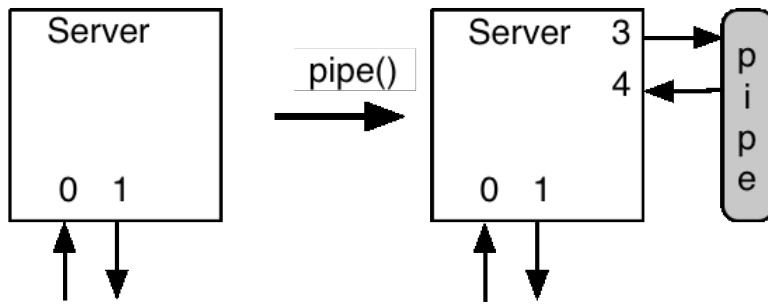
Beispiel:

`http://www.beispiel.de/cgi-bin/all-env-vars`

Steht für das Programm `all-env-vars`

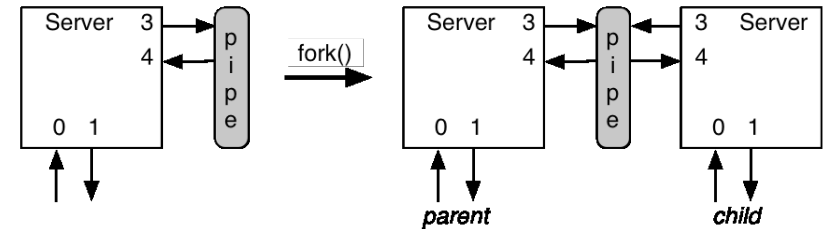
### Ablauf CGI-Programm (1)

- Webserver erzeugt eine Pipe mit `pipe()`



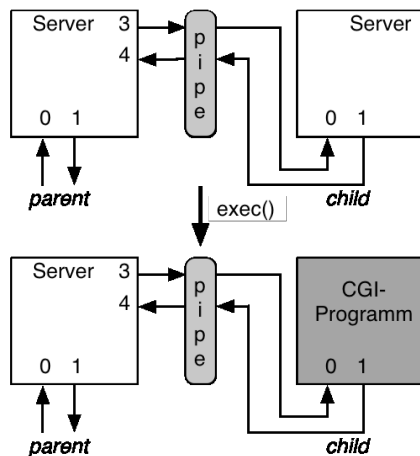
### Ablauf CGI-Programm (2)

- Kopie des Server-Prozesses mit `fork()`



### Ablauf CGI-Programm (3)

- Kind-Prozess setzt Standard-Ein/Ausgabe auf Pipe
- Aufruf des CGI-Programms mit `exec()`



### Ablauf CGI-Programm (4)

- CGI-Programm liest Standard-Eingabe und Umgebungsvariablen
- CGI-Programm gibt Ergebnis auf Standard-Ausgabe aus
- CGI-Programm beendet sich
- Server liest Ergebnis über Pipe und schickt Ergebnis zum Client

## Wichtige Umgebungsvariablen (1)

Wichtige Umgebungsvariablen:

- **SERVER\_NAME** Hostname des Servers, z. B.: `www.beispiel.de`
- **REQUEST\_METHOD** Methode des Request, `GET` oder `POST`
- **PATH\_INFO** Zusätzliche Pfad-Komponenten, z. B. `/foo/bar` für die URI `/cgi-bin/p/foo/bar`
- **QUERY\_STRING** Enthält `query`-Teil der URI bei `GET`-Anfrage
- **REMOTE\_HOST** Hostname des Clients
- **REMOTE\_ADDR** IP-Adresse des Clients

## Standard-Ein/Ausgabe

Standard-Eingabe:

- Rumpf der `POST`-Anfrage

Standard-Ausgabe:

- Enthält die berechnete Webseite
- Format: RFC-822-Header, Leerzeile, Rumpf
- Spezielle Header, die vom Server interpretiert werden

## Wichtige Umgebungsvariablen (2)

Weitere wichtige Umgebungsvariablen:

- **CONTENT\_TYPE** Typ des Rumpfes bei `POST`-Anfrage
- **CONTENT\_LENGTH** Länge des Rumpfes bei `POST`-Anfrage
- Alle übrigen Header der Anfrage mit Präfix `HTTP_` Z. B.: `HTTP_USER_AGENT`, `HTTP_HOST`

## CGI-Header

Der Server interpretiert diese Header:

- **Content-Type** MIME-Typ des Rumpfes der Nachricht
- **Status: NNN Reason** Status-Code der HTTP-Anwort
- **Location** Wenn das Argument eine URL ist, wird der Client dorthin weitergeleitet. Bei absolutem Pfad, erneute Anfrage dort.

Alle anderen Header gehen direkt in die HTTP-Anwort ein

## Einfaches CGI-Programm

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv){
    printf("Status: 200 OK\r\n");
    printf("Content-Type: text/html\r\n\r\n");

    printf("<HTML><BODY>\n");

    printf("<p>Diese Seite wurde von einem CGI-Programm erzeugt</p>\n");

    printf("<p>Deine IP (oder die deines Proxys): %s</p>\n",
        getenv("REMOTE_ADDR"));

    printf("<p>Dein Browser: %s</p>\n", getenv("HTTP_USER_AGENT"));

    printf("<a href=\"%s\">Nochmal vorbeikommen</a>\n",
        getenv("SCRIPT_NAME"));

    printf("</BODY></HTML>\n\n");
    return 0;
}
```

## FORM-Tag

**FORM**-Tag erzeugt Formular. Rumpf enthält Eingabewidgets.

Wichtige Attribute für **FORM**:

- **action** URL an die die Eingaben gesendet werden sollen
- **method** HTTP-Methode, die verwendet werden soll

Beispiel:

```
<form action="/cgi-bin/order.scm" method="POST">
... Eingabewidgets ...
</form>
```

## HTML-Formulare

*HTML-Formular* Teil eines HTML-Dokumentes, das eine Reihe von benamsten Eingabewidgets enthält.

- Benutzer füllt Formular aus
- Klick auf einen *Submit*-Button
- Browser sendet Name-Wert-Paare an den Server

## Eingabewidgets erzeugen

**INPUT** erzeugt ein Eingabewidget

Attribute:

- **type** Art des Widgets, z. B.: **text**, **password**, **radio**, **submit**, **hidden**, ...
- **name** Name des Widgets, für Referenz
- **value** Vorgabewert oder Beschriftung von Buttons

Andere Widgets: **SELECT**, **TEXTAREA**

Klick auf **submit**-Knopf sendet Formular ab

## Kodierung der Formular-Daten

Kodierung der Formular-Daten mit  
`application/x-www-form-urlencoded`:

- Name und Wert werden wie URLs kodiert
- Leerzeichen werden durch `+` repräsentiert

⇒ Ergebnis hat die Form `name1=wert1&name2=wert2&...`

Ergebnis wird gemäß Methode versendet:

- Bei `GET` als `query`-String
- Bei `POST` im Rumpf der Anfrage