

# Letzte Woche

Themen der letzten Vorlesung:

- HTTP: Methoden, Antworten, Beispiele
- Serverseitige Webprogrammierung
- Common Gateway Interface
- HTML-Formulare

# 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

# Kurzes Intermezzo: Formulare & Unicode

Kodierung mit "URL-Encoding":

- Nicht-ASCII-Zeichen werden hexadezimal kodiert:  
"%" **HEXDIG HEXDIG**
- Verfügbar: Ein Byte pro Zeichen
- Für Unicode werden bis zu vier Byte benötigt

Alternative Übertragungsmethode für Formulare:

- Verwende **POST**-Methode
- Attribut **enctype="multipart/form-data"** für **FORM**
- Kodierung des Rumpfes durch Header signalisieren

# Beispiel: HTML-Formular (1)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <body>
    <h1>Form-Test</h1>
    <p>Hallo, wie geht's?</p>
    <form action="http://www-pu[...]/cgi-bin/all-env.scm"
          method="get">
      <input type="text" name="name">
      <input type="text" name="fach">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

---

## Form-Test

Hallo, wie geht's?

<input type="text"/>	<input type="text"/>	<input type="submit" value="Submit"/>
----------------------	----------------------	---------------------------------------

## Beispiel: HTML-Formular (2)

### Form-Test

Hallo, wie geht's?

Führt zur Anfrage:

```
http://www-pu[...]/cgi-bin/all-env.scm?name=Eric+Clapton&fach=guitar
```

# Anständige CGI-Programme

Meistens überprüfen CGI-Programme zuerst ob das Formular sinnvoll ausgefüllt wurde.

Verhalten anständiger CGI-Programme im Fehlerfall:

- Formular noch einmal darstellen
- Doppelte Eingaben vermeiden (Vorgabewert nutzen)
- Fehlerstelle markieren
- Alle Fehler markieren

## cgi-form.c (1)

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

char* validate_name(char* name)
{
    if ((name != NULL) && (strlen(name) > 3))
        return name;
    else return NULL;
}

char* validate_fach(char* fach)
{
    if (fach != NULL &&
        ((strcmp(fach, "Informatik") == 0) ||
         (strcmp(fach, "Bio-Informatik") == 0)))
        return fach;
    else return NULL;
}
```

## cgi-form.c (2)

```
void do_something(char* name, char* fach){}

int main(int argc, char** argv)
{
    char* query = getenv("QUERY_STRING");
    char* name = NULL;
    char* fach = NULL;

    if (query != NULL) {
        name = query_decode(query, "name");
        fach = query_decode(query, "fach");
    }

    printf("Status: 200 OK\r\n");
    printf("Content-Type: text/html\r\n\r\n");
    printf("<HTML><BODY>\n");
}
```

## cgi-form.c (3)

```
name = validate_name(name);
fach = validate_fach(fach);

if ((name != NULL) && (fach != NULL)) {
    do_something(name, fach);
    printf("<p>Sie wurden angemeldet</p>\n");
}
else {
    printf("<FORM action=\"%s\" method=\"get\">\n",
           getenv("SCRIPT_NAME"));
    printf("<INPUT type=\"text\" value=\"%s\" name=\"name\">\n",
           (name == NULL) ? "" : name);
    printf("<INPUT type=\"text\" value=\"%s\" name=\"fach\">\n",
           (fach == NULL) ? "" : fach);
    printf("<INPUT type=\"submit\" value=\"Submit\">\n");
    printf("</FORM>\n");
}
printf("</BODY></HTML>\n\n");
return 0;
}
```

# Beobachtung

Beobachtung zu CGI-Programmen:

Ende der Ausgabe = Ende des CGI-Programms

- Das bedeutet: Der Zustand geht verloren
  - Deswegen: Zustand explizit machen
- ⇒ Explizit gemachten Zustand von Aufruf zu Aufruf transportieren

# Zustand transportieren

Einfache Methode, um Zustand zu transportieren:

- Im `query`-Teil weitergeben

Beispiel: `query.c`

- Zustand: Zähler für Anzahl der Besuche eines Benutzers

# query.c (1)

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

int main(int argc, char** argv) {

    char* query = getenv("QUERY_STRING");
    char* counter_as_charp;
    int counter;

    if (query == NULL)
        counter = 0;
    else {
        counter_as_charp = query_decode(query, "counter");
        if (counter_as_charp == NULL)
            counter = 0;
        else
            counter = atoi(counter_as_charp);
    }
}
```

## query.c (2)

```
printf("Status: 200 OK\r\n");
printf("Content-Type: text/html\r\n\r\n");

printf("<HTML><BODY>");
printf("<p>Dies ist der %d. Besuch</p>", counter);
printf("<a href=%s?counter=%d> Nochmal vorbeikommen </a>",
       getenv("SCRIPT_NAME"), counter+1);
printf("</BODY></HTML>");

return 0;
}
```

# Sessions

*Session* (Logisch zusammengehörige) Interaktion zwischen Benutzer und Web-Applikation, die sich über mehrere Anfrage/Antwort-Zyklen erstreckt.

Problem: Wie erkennt man, ob eine Anfrage zu einer bestimmten Session gehört?

Lösungsvorschläge (vorläufig):

- IP-Adresse des Clients



- IP-Adresse & Port-Nummer des Clients



- Persistente Verbindungen



# Cookies

## Lösung:

*Cookies* Speichere Session-Daten im Browser des Clients.  
Transportiere Session-Daten über HTTP-Header.

## Idee:

- **Set-Cookie**-Header in Antwort fordert den Client auf, Daten im Browser zu speichern.
- Bei jeder Anfrage des Clients: Prüfen, ob Cookie für den Server vorhanden ist. Wenn ja, dann Daten im **Cookie**-Header senden.

# Ablauf Session mit Cookies (1)

Ablauf einer Session mit Cookies:

- Server erhält Anfrage ohne `Cookie`-Header. Beginn der Session
- Server erzeugt interne Datenstruktur für Session-Daten
- Server erzeugt eindeutigen Schlüssel zur Identifikation der neuen Session, die *Session-ID*
- Server sendet Antwort mit `set-Cookie` Header, der die Session-ID enthält

## Ablauf Session mit Cookies (2)

Fortsetzung: Ablauf einer Session mit Cookies:

- Client speichert das Cookie und schickt bei allen weiteren Anfragen `Cookie-Header` mit
- Empfängt der Server eine Anfrage mit `Cookie-Header`, so können die Session-Daten über die Session-ID gefunden werden.

Alternativ: Session-Daten selber im Cookies speichern. Vorsicht: Speicherplatz begrenzt.

# Vorteile und Nachteile

## Vorteile

- Die URL enthält keinerlei Informationen über die Session
- Erprobter Mechanismus
- Programmieren mit Cookies ist einfach

## Nachteile

- Auf Unterstützung des Browsers angewiesen
- Viele Benutzer deaktivieren Cookie-Unterstützung

# Cookies im Detail

Cookies à la Netscape:

```
set-cookie      = "Set-Cookie:" cookie
cookie          = NAME "=" VALUE [ ";" set-cookie-av ]
NAME            = attr
VALUE           = value
set-cookie-av  = [ "expires" "=" value ";" " " ]
                 [ "path"   "=" value ";" " " ]
                 [ "domain" "=" value  ";" " " ]
                 [ "secure" ]
```

- **NAME:** Name des Cookies, URL-kodiert
- **VALUE:** Nutzdaten, URL-kodiert (Z. B. Session-ID)
- Mehrere unterschiedliche benannte Cookies pro Server möglich

# Cookie-Schlüsselwörter

- expires** Verfallsdatum des Cookies. Format:  
`Wdy, DD-Mon-YYYY HH:MM:SS GMT`  
Fehlt **expires**, dann gilt Cookie bis zum Beenden des Browsers
- domain** Gibt an für welche (Unter-) Domains das Cookie gesendet werden soll
- path** Client liefert Cookie nur an Ressourcen unterhalb des angegebenen Pfades aus
- secure** Bestimmt, ob Client das Cookie nur über verschlüsselte Verbindung senden darf

# Cookie-Header

Syntax des Cookie-Headers:

```
cookie          = "Cookie: "  
                cookie-value (" ; " cookie-value)*  
cookie-value    = NAME "=" VALUE  
NAME            = attr  
VALUE           = value
```

# Gültigkeitsbereiche

Beispiel: Gültigkeitsbereich von Cookies

`domain=.info.uni-tue.de`

`path=/shop/fanartikel`

Browser sendet Cookie an:

- `http://www1.info.uni-tue.de/shop/fanartikel/korb.cgi`
- `http://www2.info.uni-tue.de/shop/fanartikel/spezial/angebot.cgi`

Browser sendet Cookie **nicht** an:

- `http://shop.uni-tue.de/shop/fanartikel/korb.cgi`
- `http://www1.info.uni-tue.de/shop/diplomarbeiten.cgi`

# Cookies und Sicherheit

Cookies und Sicherheit:

Die Kontrolle, welches Cookie (und welche Daten) an den Server gesendet werden, liegt vollständig beim Client!

# Cookies ändern und löschen

## Cookie ändern:

- Server sendet `Set-Cookie` mit selbem `path`, `domain` und `NAME`, aber neuem `VALUE`.

## Cookie löschen:

- Server sendet `Set-Cookie` mit Verfallsdatum in der Vergangenheit

# Beispiel Cookies

Server sendet:

```
Set-Cookie: customer=Mandeep+Singh; path=/  
           expires=Tuesday, 15-Nov-05 23:55:00 GMT
```

In der nächsten Anfrage des Clients enthalten:

```
Cookie: customer=Mandeep+Singh
```

Server sendet:

```
Set-Cookie: product=0042; path=/
```

In der nächsten Anfrage des Clients enthalten:

```
Cookie: customer=Mandeep+Singh; product=0042
```

# Session-ID in der URL

**Alternative:** Session-ID in der URL transportieren

- Session-ID im Query-Teil oder im Pfad
- Für jeden Link: Session-ID anhängen bzw. einfügen
- Alle Seiten müssen dynamisch erzeugt werden
- Session-ID für Benutzer leicht zu ändern
- Speichern der URL wird erschwert