8.6 Sonstiges/CGI

8.6.1 Ziel

Es soll eine Anwendung (`telefonliste.c`) erstellt werden, die eine einfache Telefonliste erstellen kann. Die Oberfläche soll graphisch sein (Abbildung 1). Der Zugriff soll von überall im Netz möglich sein.

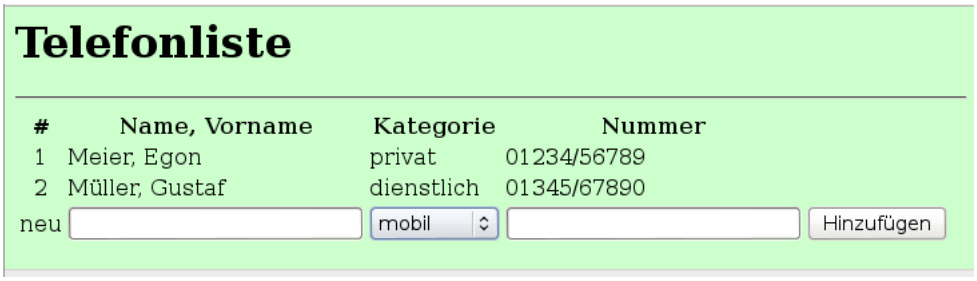

Abbildung 1: Oberfläche

8.6.2 Browser und HTML

HTML-Dokumente können mit jedem einfachen Editor erstellt werden. Es handelt sich um eine Auszeichnungssprache: Der Inhalt des Dokuments ist ein Text, der dargestellt werden soll; aber die Struktur des Dokuments ist so ähnlich wie bei einem C-Programm. Wie bei C gibt es Blöcke mit Anfang und Ende; außerdem gibt es Befehle, die die Textformatierung für diese Bereiche vorgeben. Hier ein Beispiel (`htmlbeispiel.html`):

```
1  <?xml version="1.0" ?>
2  <html>
3      <head>
4          <title>Beispiel</title>
5      </head>
6      <body bgcolor="#ddddff">
7          <h1>
8              &Uuml;berschrift
9          </h1>
10         <hr />
11         Dies ist nur ein
12         <big>
13             erstes
14         </big>
15         sch&ouml;nes Beispiel
16         f&uuml;r einen Text.
17     </body>
18 </html>
```

Mit dem Browser kann man sich die Datei ansehen, das Ergebnis zeigt Abbildung 2.

Terminal

```
heinz@debian964:~$ firefox src/htmlbeispiel.html
```

In der Adressleiste des Browser sieht man (wenn der eigene Benutzername `heinz` ist):

```
file:///home/heinz/src/htmlbeispiel.html
```

Dem absoluten Pfadnamen der Datei ist `file://` vorangesetzt, wobei `file` bedeutet, dass hier lediglich eine Datei auf dem lokalen Rechner angezeigt wird. Gibt man in der Adressleiste nur

`file:///`
an, so kann man sich durch den ganzen Rechner klicken (soweit man die entsprechenden Leserechte hat).

Mit `<xyz>` beginnt ein Bereich (man spricht von einem*element*) mit der Formatierung `xyz`, mit `</xyz>` endet er. Man spricht vom Start- und Ende-*tag* (*tag* = Markierung). Ein Bereich ohne Inhalt kann statt mit `<xyz></xyz>` auch mit `<xyz />` dargestellt werden. Darüberhinaus kann man manchem Bereich ein oder mehrere Attribute (=Eigenschaften) mitgeben, z.B.:
`<witz niveau="flach">...</witz>`
Durch diese Attribute kann die Formatierung modifiziert werden.

Bei einem HTML-Dokument ist das äußerste Element `<html>`, darin befinden sich `<head>` und `<body>`. Was in `<head>` steht, beeinflusst das gesamte Dokument (hier z.B. die Anzeige in der Titelleiste), was auch immer in `<body>` steht, beeinflusst nur die Stelle, an der es steht.

Weitere Informationen zu HTML findet man im Web unter SELFHTML sowie hier im Kapitel X1.

HTMl ist vorwiegend für die Darstellung auf einem Computer gedacht. Für die Erstellung von Texten auf Papier gibt es andere Auszeichnungssprachen wie z.B. LaTeX und SGML.

8.6.3 Browser, Server und HTML-Dokumente

8.6.3.1 Theorie Bisher konnte das Dokument nur auf dem eigenen Rechner angezeigt werden. Wie aber funktioniert es, wenn man Seiten auf entfernten Rechnern aufruft?

Wenn man in seinem PC einen Browser öffnet und dort in die Adressleiste:
`http://www.csbme.de/index.html`
eingibt, liefert der Rechner mit dem DNS-Namen `www.csbme.de` die Datei mit dem Dateinamen `index.html` an den PC, und der Browser stellt sie in seinem Ergebnisfenster dar (Abbildung 3)[1].

Der Browser ist derjenige, der die Verbindung öffnet; er wird Client genannt. Das entsprechende Programm auf dem entfernten Rechner, das diese Anfrage bedient, nennt man Server, in diesem Fall ein Webserver. Ein häufig benutztes Webserver-Programm heißt Apache.

Das benutzte Protokoll für den Transport von Webseiten heißt HTTP (*hypertext transport protokoll*). Der Ablauf einer Transaktion ist sehr einfach: Der Client stellt eine Anfrage (*request*, worauf der Server die Antwort sendet (*response*).

Beide – Anfrage und Antwort – sind reine Textnachrichten in ASCII (also ohne Umlaute), bestehend aus Zeilen, die durch `\n` getrennt sind. Die Textnachrichten bestehen aus einem Kopf (mit mindestens einer Zeile) und einem Rumpf (kann auch leer sein). Kopf und Rumpf sind durch eine Leerzeile getrennt.

8.6.3.2 Wie sieht eine Anfrage aus? Hier ist ein Beispiel für eine Anfrage:

[1] Da HTTP das Standardprotokoll eines Browsers ist, kann man den Teil `http://` weglassen. Ebenso gibt es ein Standarddokument `index.html` für jedes Verzeichnis, so dass dieser Teil auch wegfallen kann.

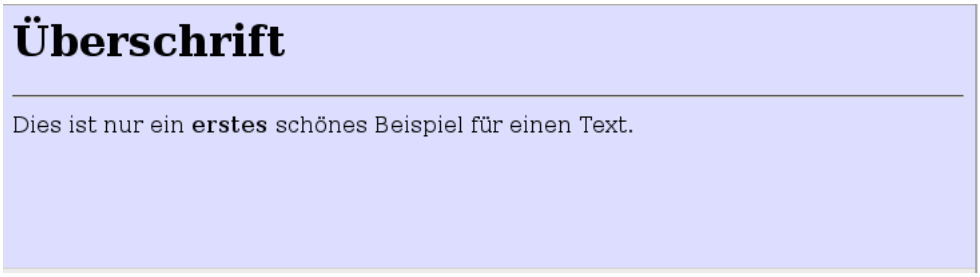

Abbildung 2: htmlbeispiel.html im Browser

Terminal
```
GET /index.html HTTP/1.0
Accept-Language: de-de

```

Die erste Zeile besteht aus der Methode (`GET`), dem Pfadnamen der gewünschten Datei auf den Zielrechner (`/index.html`) und der Protokoll-Version (`HTTP/1.0`). Danach folgt eine optionale Headerzeile, bestehend aus Name einer Eigenschaft (hier z.B. die akzeptierten Sprachen), Doppelpunkt, Leerzeichen und Inhalt dieser Eigenschaft (hier z.B. deutsch). Zum Schluss findet man eine Leerzeile, mit der der Header abgeschlossen wird. Der Rumpf ist in diesem Fall leer: Es muss nichts weiter an den Server übertragen werden.

8.6.3.3 Wie sieht eine Antwort aus?

Und hier ein Beispiel für die Antwort:

Terminal
```
HTTP/1.0 200 OK
Content-Type: text/html
Server: Apache/1.3.5

<html>
  <head></head>
  <body>
    Es ist genau 10:12:35.
  </body>
</html>
```

Die erste Zeile besteht aus Protokollversion (`HTTP/1.0`), Antwortcode (`200` heißt Erfolg) und einer für den menschlichen Leser gedachten Erklärung dieses Codes. Es folgen zwei Antwortzeilen. Die Zeile mit `Content-Type` ist wichtig, denn sie sagt dem Client, welches Format die Antwort hat:

- `text/plain` ist einfacher, unformatierter ASCII-Text
- `text/html` ist ein HTML-Dokument
- `audio/mpeg` sind codierte Audiodaten
- `image/jpeg` sind Bilder im JPEG-Format

Man spricht bei diesen Formatbezeichnungen von MIME-Typen[2]

[2]MIME ist die Abkürzung für *multimedia mail extension* und benennt einen Standard, der für die Übertragung von beliebigen Daten über E-Mail entwickelt wurde. Wie HTTP ist E-Mail nämlich auch rein ASCII-basiert und

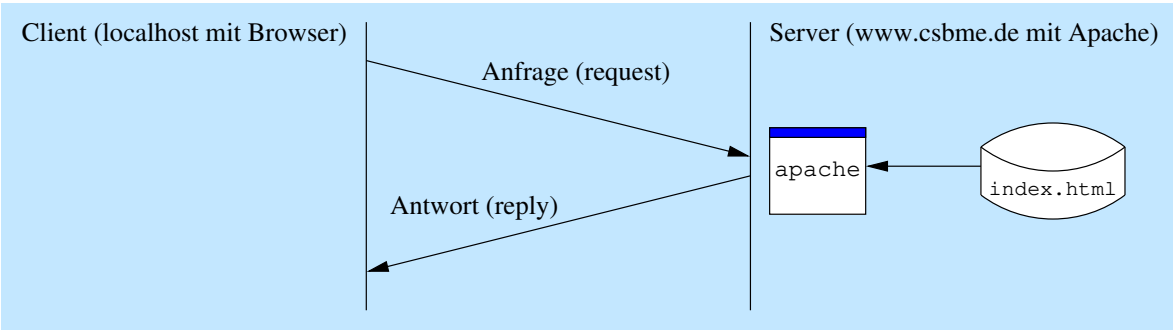

Abbildung 3: HTTP-Transaktion

8.6.3.4 Praxis-Beispiel Das HTTP-Protokoll kann man auch von Hand nachstellen, indem man mit einem geeigneten Client Zeichen für Zeichen eingibt und abschickt. Dazu eignen sich der Telnet-Client `telnet` sowie das Programm `netcat` (muss ggf. nachinstalliert werden):

Terminal

```
heinz@debian964:~$ netcat -C savannah.nongnu.org 80
GET / HTTP/1.0
(Leerzeile)
HTTP/1.1 200 OK
Date: Thu, 06 Mar 2014 08:30:14 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze18
Connection: close
Content-Type: text/html; charset=utf-8

<?xml version="1.0" encoding="utf-8"?>
.....
</body>
</html>
```

Hier wird also eine Verbindung zum Rechner `savannah.nongnu.org` an Port 80 hergestellt:

- `savannah.nongnu.org` ist der DNS-Name des Zielrechners. Anstelle des Namens hätte man auch die IP-Adresse 140.186.70.71 verwenden können[3].
- `80` ist die Portnummer. Damit an einem Rechner mehrere Server unterschiedliche Aufgaben übernehmen können, gibt es zusätzlich zur IP-Adresse immer eine Portnummer. Während die IP-Adresse zur Adressierung zwischen verschiedenen Rechnern gedacht ist, kann man mit der Portnummer innerhalb eines Rechners das richtige Serverprogramm auswählen. Die Portnummer für den Web-Server ist meistens 80. Genauso haben die anderen bekannten Server-Arten (Proxy, E-Mail, ftp, ssh und andere) ebenfalls eine übliche Portnummer. Auf Linux-Systemen findet man eine Übersicht in der Datei `/etc/services`.

8.6.3.5 Installation eines eigenen Webservers Das Webserver-Programm Apache ist sehr leistungsfähig und hat weltweit einen sehr hohen Marktanteil. Darüberhinaus ist es kostenlos. Deshalb lohnt sich die hier beschriebene Installation[4][5]:

Terminal

```
heinz@debian964:~$ su          # Administrator werden
Passwort:       # hier Admin.-Passwort eingeben
root@debian964:~# apt-get install apache2
...
root@debian964:~# exit
```

kann eigentlich nur ASCII-Zeichen zwischen 32 und 127 problemlos transportieren. Für alle anderen zu transportierenden Bytes musste daher eine Lösung erstellt werden, die – ähnlich wie UTF-8 – die restlichen möglichen Bitkombinationen eines Bytes kodiert. Auf Linux-Systemen findet man in der Datei `/etc/mime.types` weitere MIME-Typen. Auch der Browser kann seine MIME-Typen anzeigen. Die zweite Spalte mit den Dateiendungen wird dabei nicht für HTTP gebraucht – dort wird der Content-Type alias MIME-Type ja mitgesendet – sondern für die lokale Anzeige von Dateien mit `file:///`.

[3] Dabei handelt es sich um eine 32-Bit-Zahl. Die übliche Darstellung erhält man, indem man sie in vier Bytes aufteilt und jedes Byte einzeln als Dezimalzahl aufschreibt. Zwischen die vier Dezimalzahlen kommen zur Trennung Punkte – fertig. In Wirklichkeit lautet die IP-Adresse also 0x8CBA4647 oder (binär) 1000 1100 1011 1010 0100 0110 0100 0111.

[4] Wer Ubuntu benutzt oder Debian ohne eigenes Administrator-Konto, der kann den Befehl `su` nicht benutzen, weil es bei ihm kein Administrator-Passwort gibt. Stattdessen gibt er `sudo bash` ein und anschließend sein eigenes Passwort.

[5] Am Ende der Zeilen befinden sich Anmerkungen (hinter dem Raute-Zeichen); sie brauchen nicht eingegeben zu werden.

Jetzt kann man im Browser eingeben:
`http://localhost/index.html`
Mit `localhost` ist stets der eigene Rechner gemeint. Er kann ebenso über die Adresse 127.0.0.1 (oder 127.0.0.2 usw.) angesprochen werden[6].

Dann sollte man die Startseite des Apache-Servers sehen mit der Nachricht „It works“.

8.6.3.6 Erstes eigenes Dokument Man kann jetzt als Administrator `root` ein eigenes Dokument erstellen und ablegen. Dazu muss man wissen, dass jeder Web-Server ein eigenes Stammverzeichnis für seine Dokumente hat (so genannte *document root*): Dieses Verzeichnis liegt beim Apache standardmäßig unter:
`/var/www`
Im Browser erscheint es als:
`http://rechnername/`
Nun kann das Dokument angelegt werden:

Terminal
```
heinz@debian964:~$ su
Passwort:
root@debian964:~# gedit /var/www/eins.html
root@debian964:~# exit
```

Danach sollte unter der Adresse
`http://localhost/eins.html`
die angelegte Datei zu sehen sein.

8.6.3.7 Dokumentenverzeichnis für normalen Nutzer einrichten Im Verzeichnis `/var/www/` hat der normale Benutzer (hier `heinz` genannt, bitte eigenen Anmeldenamen einsetzen) kein Schreibrecht. Daher ist es sinnvoll, ein Unterverzeichnis `fet1u` anzulegen und ihm dafür das Schreibrecht zu geben, indem man ihn zum Eigentümer erklärt. Zum Schluss wird eine Verknüpfung erstellt, damit `heinz` das neue Verzeichnis auch unter dem Namen `www` in seinem persönlichen Verzeichnis finden kann (kleine Vereinfachung).

Terminal
```
heinz@debian964:~$ su
Passwort:
root@debian964:~# mkdir /var/www/fet1u                  # make direct.
root@debian964:~# chown heinz /var/www/fet1u            # change owner
root@debian964:~# ln -s /var/www/fet1u /home/heinz/www # symlink anl.
root@debian964:~# exit
```

Nun kann `heinz` das Dokument `zwei.html` in `/home/heinz/www` anlegen:

Terminal
```
heinz@debian964:~$ cd              # ins persönliche Verz.
heinz@debian964:~$ cd www          # change directory
heinz@debian964:~$ gedit zwei.html # Editieren
```

Jetzt kann die Seite im Browser aufgerufen werden:
`http://localhost/fet1u/zwei.html`

8.6.4 CGI ohne GET und POST

8.6.4.1 Theorie Die bisherigen HTML-Dokumente sind unveränderlich (= statisch) und können beim Server und beim Client nichts bewirken (= passiv). Mehr Möglichkeiten erhält man der CGI-Schnittstelle der Web-Server. Sie gehört ursprünglich nicht zum HTTP-Protokoll, war aber bereits in frühen Web-Servern vorhanden und ist nun seit Jahren etablierter Standard.

[6] Wenn einen jemand einlädt, 127.0.0.1 zu hacken und zu zerstören, sollte man eher vorsichtig werden.

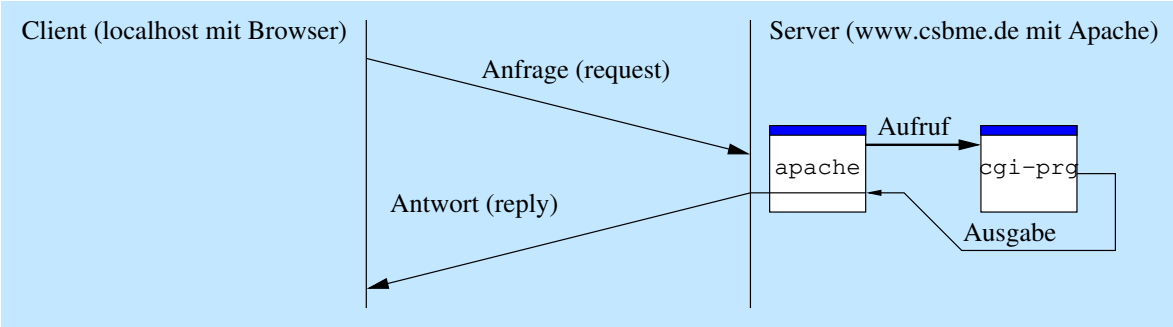

Abbildung 4: HTTP mit CGI

Ein CGI-Programm (auch CGI-Skript genannt, weil CGI-Programme oft in Skriptsprachen realisiert werden) ist ein ganz normales Programm. Aber es wird nicht vom Benutzer aufgerufen, und seine Textausgabe landet nicht auf dem Bildschirm. Stattdessen wird es vom Web-Server *aufgerufen*, und seine Textausgabe wird an den Client *weitergeleitet* (siehe Abbildung 4).

8.6.4.2 Praxis-Beispiel Apache erkennt CGI-Programme nicht an einer Endung (wie z.B. `cgi`), sondern an ihrem Ort im Verzeichnisbaum. CGI-Programme liegen in einem eigenen Stammverzeichnis (*CGI root*) mit dem absoluten Pfad:
`/usr/lib/cgi-bin`
Im Browser erscheint dieser Pfad jedoch als:
`http://rechnername/cgi-bin`
Das erste CGI-Programm soll nur „Hallo“ auf das Browserfenster ausgeben. Es hat folgenden Quelltext:

```
1 #include <stdio.h>
2 int main(void)
3 {
4    printf("Content-Type: text/plain\n"); /* Header-Zeile       */
5    printf("\n");                         /* Leerzeile          */
6    printf("Hallo\n");                    /* Ausgabe an Client */
7    return 0;                     /* An Web-Server Erfolg melden */
8 }
```

Als MIME-Type wurde hier `text/plain` gewählt, um kein langes HTML-Dokument erzeugen zu müssen. `text/html` ist genauso möglich.

Terminal

```
heinz@debian964:~$ su                    # Administrator werden
Passwort:
root@debian964:~# cd /usr/lib/cgi-bin # Verzeichnis wechseln
root@debian964:~# gedit eins.c        # Editieren
root@debian964:~# gcc -o eins eins.c  # Compilieren,Ausg.-Datei:eins
root@debian964:~# ./eins              # Probeweise aufrufen von der
                        # Kommandozeile
Content-Type: text/plain

Hallo
root@debian964:~# exit
```

Man sieht bei der Ausgabe die Headerzeile (die einzige erforderliche) und die Leerzeile. Sie sind wichtig. Ohne diese Zutaten erhält man nur die Meldung „500 Server Error“. Es fehlt jedoch

die Zeile mit dem Statuscode („HTTP/1.0 200 OK“). Sie ist nicht nötig, da sie vom Web-Server automatisch hinzugefügt wird.

Jetzt kann das Programm im Browser aufgerufen werden unter:

`http://localhost/cgi-bin/eins`

Dann sollte das „Hallo“ im Browserfenster erscheinen.

8.6.4.3 CGI-Verzeichnis für normalen Nutzer einrichten Auch hier sollte man ein Verzeichnis einrichten, in dem der normale Benutzer Schreibrecht hat. Also wird auch hier ein Unterverzeichnis `fet1u` angelegt, `heinz` wird zum Eigentümer ernannt und ein Link ins persönliche Verzeichnis von `heinz` gelegt:

Terminal

```
heinz@debian964:~$ su
Passwort:
root@debian964:~# mkdir /usr/lib/cgi-bin/fet1u
root@debian964:~# chown heinz /usr/lib/cgi-bin/fet1u
root@debian964:~# ln -s /usr/lib/cgi-bin/fet1u /home/heinz/cgi
root@debian964:~# exit
```

Jetzt kann `heinz` wiederum ein Programm `zwei.c` in `/home/heinz/cgi` anlegen. Es soll denselben Text ausgeben wie `eins.c`, jedoch zur Abwechslung in HTML:

```
1  #include <stdio.h>
2  #include <time.h>
3  int main(void)
4  {
5     time_t t;
6     t=time(0);
7     printf("Content-Type: text/html\n");   /* Header-Zeile        */
8     printf("\n");                          /* Leerzeile           */
9     printf("<html><head></head>\n"
10           "  <body bgcolor=\"#ffffcc\">\n"
11           "    <h1>");
12    printf("    %s", ctime(&t));
13    printf("    </h1>\n"
14           "  </body>\n"
15           "</html>\n");
16    return 0;                             /* Erfolg melden       */
17 }
```

Terminal

```
heinz@debian964:~$ cd
heinz@debian964:~$ cd cgi
heinz@debian964:~$ gedit zwei.c
heinz@debian964:~$ gcc -o zwei zwei.c
heinz@debian964:~$ ./zwei                    # Test auf Kommandozeile
Content-Type: text/html

<html><head></head>
  <body bgcolor="#ffffcc">
    <h1>Thu Mar  6 11:56:31 2014</h1>
  </body>
</html>
heinz@debian964:~$ ./zwei > zweiaus.html #optional:Erzeuge Ausgabedat.
heinz@debian964:~$ firefox zweiaus.html  #Test d.Ausgabedat.i.Browser
```

Wenn alles geklappt hat, kann man das Skript vom Browser aus aufrufen mit:
`http://localhost/cgi-bin/fet1u/zwei`

8.6.5 CGI mit GET-Methode

8.6.5.1 Theorie Bis jetzt konnte der Client ein CGI-Programm nur aufrufen und Informationen holen. Es fehlt aber noch die Möglichkeit, Informationen an das CGI-Programm zu schicken. Dazu gibt es mehrere Methoden; hier soll die GET-Methode beschrieben werden.

Bei der GET-Methode wird die zu schickende Information an die URL (die Adresszeile im Browser) angehängt:
`http://localhost/cgi-bin/zwei?zusatzkram`
Das CGI-Programm erhält diese Information in einer so genannten Umgebungsvariable. Die bekannten Betriebssysteme (Linux, MacOS, Windows) stellen jedem Programm eine Reihe von Umgebungsvariablen zur Verfügung, die je nach Wunsch dem Programm bestimmte Informationen zur Verfüngung stellen sollen. Mit dem Befehl `env` unter Linux sowie `set` unter Windows kann man sich diese Variablen anzeigen lassen:

Terminal
```
heinz@debian964:~$ env                    # alle anzeigen
...
heinz@debian964:~$ export WETTER="gut"  # setze Wetter auf "gut"
heinz@debian964:~$ echo "Heute: $WETTER"# Ausgabe
Heute: gut
heinz@debian964:~$ cal                    # Sprache: deutsch
     März 2014
So Mo Di Mi Do Fr Sa
...
heinz@debian964:~$ export LANG=POSIX    # setze Sprache auf
heinz@debian964:~$ cal                    # International
     March 2014
Su Mo Tu We Th Fr Sa
...
heinz@debian964:~$ LANG=de_DE.utf8 date # fuehre date aus mit LANG=...
```

Die Umgebungsvariable für CGI-Programme heißt `QUERY_STRING`. Sie wird durch den Web-Server auf die Zeichenkette gesetzt, die in der Adresszeile hinter dem Fragezeichen beginnt, im oben genannten Fall hat sie den Inhalt `"zusatzkram"` (ohne Anführungszeichen). Im Programm kann sie durch Aufruf der Funktion `getenv()` geholt werden (definiert in `<stdlib.h>`):

```
1    char *p;
2    p=getenv("QUERY_STRING");
```

8.6.5.2 Praxisbeispiel Das folgende Beispielprogramm zeigt den `QUERY_STRING` an:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void)
4  {
5      char *s;
6      s=getenv("QUERY_STRING");
7      if(s==NULL)
8          s="(leer)";
9      printf("Content-Type: text/plain\n\n");
10     printf("Ergebnis: >>%s<<\n", s);
11     return 0;
```

```
12 }
```

Terminal

```
heinz@debian964:~$ cd; cd cgi
heinz@debian964:~$ gedit getdemo.c
heinz@debian964:~$ gcc -o getdemo getdemo.c
heinz@debian964:~$ ./getdemo                    # ohne QS, v.d.Konsole
Content-Type: text/plain

Ergebnis: >>(leer)<<
heinz@debian964:~$ QUERY_STRING=blubb ./getdemo # mit QS, v.d.Konsole
Content-Type: text/plain

Ergebnis: >>blubb<<
heinz@debian964:~$ netcat -C localhost 80       # mit QS, vom Netz
GET /cgi-bin/kurs/getantwort?123 HTTP/1.0
(Leerzeile)
HTTP/1.1 200 OK
Date: Thu, 06 Mar 2014 12:19:29 GMT
...
Content-Type: text/plain

Ergebnis: >>123<<
```

Nun kann der Einsatz im Browser erprobt werden:
`http://localhost/cgi-bin/fet1u/getdemo?quark`

8.6.6 HTML-Formulare und CGI mit GET

Hier ist ein einfaches Beispiel für ein HTML-Formular (`src/simpleformget.html`):

```
1 <html>
2  <head></head>
3  <body>
4   <form>
5    <input type="text" name="eingabe" />
6    <input type="submit" />
7   </form>
8  </body>
9 </html>
```

Gibt man in das Feld das Wort „Hallo“ ein und schickt das Formular ab, erscheint in der Adressleiste folgende URL:
`simpelformget.html?eingabe=Hallo`
Der Ergebnisstring muss also aufgeteilt werden in den Namen der Variablen (vor dem Gleichheitszeichen) und in ihren Inhalt (dahinter). Nach Eingabe von „Bad Salzuflen-Schötmar“ erscheint dort:
`...html?eingabe=Bad+Salzuflen-Sch%F6tmar`
Leerzeichen werden also durch Pluszeichen maskiert, andere Satz- und Sonderzeichen durch ein Prozent und die Hexadezimaldarstellung der ASCII- (bzw. ISO-Latin-) Position des Zeichens. Bei Zeichen, die im Unicode eine Position oberhalb von 255 haben, wird es etwas komplizierter: Das Ω-Zeichen hat die Position 8446; in HTML wird es als `&#8446;` dargestellt; hier ergibt es: `%26%238446%3B`, wobei `%26` das &-Zeichen darstellt, `%23` das #-Zeichen und `%3B` das Semikolon.

In `src/simpleformget2.html` wurde das HTML-Formular um ein weiteres Feld `eingabeB` ergänzt. Gibt man links L ein und rechts R, erhält man:

`eingabe=L&eingabeB=R`

Der Ergebnisstring muss also außerdem aufgeteilt werden in die Ergebnisse der einzelnen Eingabefelder.

8.6.7 Vereinfachung durch cgilib

Das oben genannte Ergebnis muss nun zerteilt werden.

- Die Felder sind voneinander durch & getrennt.
- Name und Inhalt eines Feldes sind durch ein Gleichheitszeichen getrennt.

Man kann diese Aufgabe von Hand erledigen. Dazu braucht man aber viele Programmzeilen in C (z.B. in `src/zerteil1.c`). Es gibt aber auch C-Funktionsbibliotheken, die einem das abnehmen können. Nebenbei erledigen sie zudem die Dekodierung der Leer-, Satz- und nicht-ASCII-Sonderzeichen. Hier soll die Bibliothek `cgilib` verwendet werden, weil sie einfach zu benutzen ist[7]. Hier braucht man nur die drei Funktionen `cgiInit()`, `cgiGetValue()` und `cgiFree()`. Wie man in `src/zerteil2.c` sieht, werden sie ganz ähnlich benutzt wie `fopen()`, `fscanf()` und `fclose()`:

```
1  #include <stdio.h>
2  #include <cgi.h>
3  int main(void)
4  {
5      char *inhalt;
6      s_cgi *cgi_merker;
7
8      printf("Content-Type: text/plain\n\n");
9      cgi_merker = cgiInit();
10     if(cgi_merker==NULL)
11     {
12         printf("CGI-Fehler!\n");
13         return 0;
14     }
15     inhalt=cgiGetValue(cgi_merker, "EingabeA");
16     if(inhalt!=NULL)
17         printf("Inhalt von EingabeA: %s\n", inhalt);
18     inhalt=cgiGetValue(cgi_merker, "EingabeB");
19     if(inhalt!=NULL)
20         printf("Inhalt von EingabeB: %s\n", inhalt);
21     cgiFree(cgi_merker);
22     return 0;
23 }
```

8.6.8 Speicherung in Dateien

Nun soll das CGI-Programm die bisherigen Eingaben in einer Datei speichern können[8]. Dazu braucht der Server (Login-Name: `www-data`) in einem Verzeichnis auf dem Rechner die Schreibberechtigung[9]. Also legt man ein solches Verzeichnis an und vergibt die Berechtigung[10]:

[7] Außerdem gibt es noch `libcgic2` und `uncgi`

[8] Eine Alternative wäre die Speicherung in einer Datenbank (z.B. mySQL).

[9] Wenn man nicht weiß, welchen Login-Namen der Server-Prozess auf dem jeweiligen System hat, kann man dessen User-ID über ein CGI-Programm herausfinden. Es ist unter `src/uid.c` verfügbar. Die Ausgabe des Programms (im Browser oder per `netcat` ist genau diese ID. Danach sucht man in der Nutzerdatei `/etc/passwd` nach dem Login-Namen (1. Spalte) des Nutzers mit der ermittelten User-ID (3. Spalte).

[10] Eigentlich ist es nicht schön, unterhalb des `cgi-bin`-Verzeichnisses ein für alle schreibbares Verzeichnis anzulegen; hier passiert es der Einfachheit halber.

Terminal

```
heinz@debian964:~$ mkdir /home/heinz/cgi/schreib  #Verz. anlegen
heinz@debian964:~$ su
Passwort:
root@debian964:~# chown www-data /home/heinz/cgi/schreib #Eigentuemer
root@debian964:~# exit
```

Danach kann das CGI-Programm wie jedes Programm dort Daten ablegen (`src/dateischreiben.c`):

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <errno.h>
4  int main(void)
5  {
6     FILE *dateimerker;
7     printf("Content-Type: text/plain\n\n");
8     dateimerker=fopen(
9     "/usr/lib/cgi-bin/fet1u/schreib/neu.txt","w");
10    if(!dateimerker)
11    {
12       printf("fopen():%s\n", strerror(errno));
13       return 0;
14    }
15    printf("o.k.\n");
16    fprintf(dateimerker, "xyz");
17    fclose(dateimerker);
18    return 0;
19 }
```

Genauso kann es die dort abgelegten Daten auch wieder lesen. Somit kann, wenn man das will, jeder Prozess alles lesen, was dort jemals abgelegt worden ist.

8.6.9 Mögliche Struktur: HTML-Datei und CGI-Programm getrennt

Abbildung 5: Statisches Formular `sg1eineben.html`

In einem simplen Gästebuch soll jede bisherige Texteingabe angezeigt werden. Für das statische Formular (Abbildung 5) dient `sg1eingeben.html` (im Verzeichnis `/home/heinz/www`):

```
1  <html>
2   <head><title>Bitte etwas eingeben</title></head>
3   <body bgcolor="#ffffcc">
4    Neue Eingabe:
5    <form action="http://localhost/cgi-bin/fet1u/sg1anzeigen">
6     <input type="text" name="eingabe" />
7     <input type="submit" />
8    </form>
9   </body>
10 </html>
```

`sg1anzeigen.c` (im Verzeichnis `/home/heinz/cgi`) liest und schreibt die Datei und erzeugt die Ausgabe (Abbildung 6) und den Link zurück zum Formular:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#define DATEINAME "schreib/sg1daten.txt"
int main(void)
{
    int c;
    FILE *dm;
    char *p;
                                                /* 1) QUERY_STRING holen */
    p=getenv("QUERY_STRING"); if(!p) p="";
                                                /* 2) an Datei anhaengen */
    dm=fopen(DATEINAME,"a");
    if(!dm)
    {
        printf("fopen(%s): %s\n", strerror(errno));
        return 0;
    }
    fprintf(dm, "%s\n", p);
    fclose(dm);
                                               /* 3) HTML-Header ausgeben */
    printf("Content-Type: text/html\n\n"
           "<html>\n"
           " <head><title>Bisher eingegeben:</title></head>\n"
           " <body bgcolor=\"#ccffff\">\n"
           "  Bisher eingegeben: <pre>\n");
                                               /* 4) Datei lesen und ausg. */
    dm=fopen(DATEINAME,"r");
    if(!dm)
    {
        printf("fopen(): %s\n", strerror(errno));
        return 0;
    }
    while((c=fgetc(dm))!=EOF)
        putchar(c);
    fclose(dm);
                                              /* 5) HTML-Ausgabe beenden */
    printf("  </pre>\n"
           "  <a href=\"/fet1u/sg1eingeben.html\">zur&uuml;ck</a>\n"
           " </body>\n"
           "</html>\n");
```

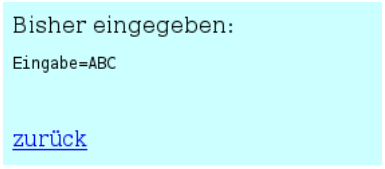

Abbildung 6: Dynamische Ausgabe mit `sg1anzeigen`

```
43      return 0;
44  }
```

Man sieht, dass das CGI-Programm hier eine Reihe von Aufgaben hat:

- Entgegennehmen der Eingabe (besser mit `cgilib`)
- Schreiben der Datei (zum Anhängen) und schließen
- Lesen der Datei (und schließen)
- Sofortige HTML-Ausgabe
- mit Link zum Eingabeformular am Ende der Ausgabe

8.6.10 Mögliche Struktur: Ein CGI-Programm für alles

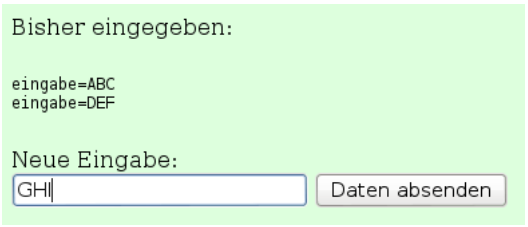

Abbildung 7: Dynamische Ausgabe und Formular auf einer Seite bei `sg2alles`

Will man Formular und CGI-Ausgabe auf einer Seite haben, kann man das Formular mit in das CGI-Programm packen[11] (Abbildung 7).

Dazu kann man das bisherige Programm `sg1anzeigen.c` nehmen; man muss lediglich die Ausgabe des HTML-Formulars am Schluss des Programms anhängen. Da Ergebnis sieht man in `src/sg2alles.c`.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <errno.h>
5  //#define DATEINAME "/usr/lib/cgi-bin/fet1u/schreib/sg1daten.txt"
6  #define DATEINAME "schreib/sg1daten.txt"
7  int main(void)
8  {
9      int c;
10     FILE *dm;
11     char *p;
12                                          /* 1) QUERY_STRING holen */
13     p=getenv("QUERY_STRING"); if(!p) p="";
14                                          /* 2) an Datei anhaengen */
15     dm=fopen(DATEINAME,"a");
16     if(!dm)
17     {
18         printf("fopen(%s): %s\n", strerror(errno));
19         return 0;
```

[11] Es gibt auch noch andere Möglichkeiten, z.B. die Verwendung von Frames und I-Frames; dabei kann ein Teil des Fensters für eine CGI-Ausgabe reserviert werden, während in einem anderen Teil die Eingabefelder sichtbar bleiben.

```
20      }
21      fprintf(dm, "%s\n", p);
22      fclose(dm);
23                                              /* 3) HTML-Header ausgeben */
24      printf("Content-Type: text/html\n\n"
25             "<html>"
26             "<head><title>G&auml;stebuch</title></head>"
27             "<body bgcolor=\"#ddffdd\">\n"
28             "Bisher eingegeben:<pre>\n");
29                                              /* 4) Datei lesen und ausgeben */
30      dm=fopen(DATEINAME, "r");
31      if(!dm)
32      {
33          printf("fopen(): %s\n", strerror(errno));
34          return 0;
35      }
36      while((c=fgetc(dm))!=EOF)
37          putchar(c);
38      fclose(dm);
39                                              /* 5) HTML-Ausgabe beenden */
40      printf("</pre>\n"
41             "Neue Eingabe:\n"
42             "<form>\n"
43             "<input type=\"text\" name=\"eingabe\" />\n"
44             "<input type=\"submit\" />\n"
45             "</form>\n"
46             "</body>\n"
47             "</html>\n");
48      return 0;
49  }
```

Hier fällt auf, dass im `<form>`-Tag das `action`-Attribut fehlt. Das liegt daran, dass der Default-Wert für dieses Attribut immer das eigene Dokument ist, und genau das wird ja beim Klicken auf den Abschicken-Knopf erneut aufgerufen.

8.6.11 CGI mit POST-Methode

Bei der POST-Methode wird die Information vom Client zum Server nicht in der URL übergeben, sondern im Rumpf der Anfrage. Damit der Server weiß, auf wie viele Bytes er warten soll, erhält er eine Headerzeile mit dem Namen `Content-Length`. Der Server liest die bestimmte Anzahl an Bytes und gibt sie auf die Standardeingabe des CGI-Programms. Dieses kann die Eingabe dann mit `scanf()` oder `getchar()` so lesen, als wenn sie von der Tastatur kämen:

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   int main(void)
4   {
5       int c;
6       printf("Content-Type: text/plain\n\n");
7       printf("Ergebnis: >>");
8       c=getchar();
9       while(c!=EOF)
10      {
11          putchar(c);
12          c=getchar();
```

```
13      }
14      printf("<<\n");
15      return 0;
16  }
```

Terminal

```
heinz@debian964:~$ netcat -C localhost 80
POST /cgi-bin/kurs/putantwort HTTP/1.0
Content-Length: 20
(Leerzeile)
01234567890123456789012345
HTTP/1.1 200 OK
Date: Thu, 06 Mar 2014 12:28:37 GMT
...
Content-Length: 32
...
Content-Type: text/plain

Ergebnis: >>01234567890123456789<<
```

Die POST-Methode hat Vorteile:

- Der normale Nutzer kann nicht auf der Browser-Adresszeile spicken, welche Daten er an den Client übermittelt (mit kleinen Tricks geht es schon).
- Die Eingabelänge ist prinzipiell unbegrenzt.

Der Nachteil besteht vor allem darin, dass man als Programmierer nicht mehr einfach anhand der Browser-Adresszeile sehen kann, wie die Formulareingabe übertragen wird.

Der Umstieg auf die POST-Methode ist einfach: Wenn man `cgilib` benutzt, reicht es aus, im `<form>`-Tag das Attribut `"method=POST"` zu setzen. `cgilib` kann mit beiden Methoden gleichermaßen umgehen.