Praktikum Verteilte Systeme

Das Praktikum findet im Pool-Raum SI0024 des Laborbereichs Technische Informatik statt. Für das Praktikum ist eine Vorbereitung erforderlich. Während des Praktikums ist ein Praktikumsprotokoll mit Namen, E-Mail-Adresse, Datum und allen wichtigen Ergebnissen zu erstellen und mit dem Betreuer am Bildschirm durchzusprechen. Das Protokoll soll für 2.1 und 2.2 die für einen Nachweis der Funktion erforderlichen Ein- und Ausgaben enthalten. Kopieren Sie diese (als Text!) aus der Konsole ins Protokoll und beantworten Sie außerdem die Fragen zu 2.3. Laden Sie das Protokoll zusammen mit dem gezippten Source-Code im OSCA-Dokumentenabgabeordner unter praktxx_Name1_Name2.docx hoch (xx=Nr. der Praktikumsaufgabe).

Praktikumsaufgabe 3 (Minimaler Web-Server)

Ziel der Aufgabe ist es, auf Basis eines TCP-Servers (tcp_server.c aus kap4beispiele.zip) einen minimalen Web-Server in der Programmiersprache C zu realisieren. Die Aufgabe muss bis in der Woche ab dem 23.03.2020 bis zum Ende des jeweiligen Praktikumstermins zu 80% fertiggestellt werden.

Ihr Programm muss eine Teilmenge des HTTP-Protokolls implementieren. Dieses wird im späteren Verlauf der Vorlesung noch detaillierter vorgestellt werden. Für die hier anstehende Übung sind folgende Angaben nützlich:

- HTTP baut als Protokoll der Anwendungsschicht auf TCP auf.
- HTTP unterscheidet folgende zwischen Client und Server ausgetauschte Nachrichten:
  - vom Client initiierte *Requests* (Anfragen) und
  - die *Responses* (Antworten) vom Server.
- Eine Nachricht besteht aus Nachrichten-Kopf (*Header*) und Nachrichten-Rumpf (*Body*).
- Durch entsprechende Attribute im Header können verschiedene *Request-Methoden* definiert werden. **GET**-Requests sind die häufigste Zugriffsmethode. Mit **GET** werden Daten, zumeist eine als URI (*Uniform Resource Identifier*) eindeutig definierte Ressource, an den Server übertragen. URIs sind meist auf 255 Bytes beschränkt.
- Ein Request wird durch den Server mit einem sog. Status-Code beantwortet. Dieser wird im Header (der ersten Zeile) der Response gesetzt. Die Response hat das Format:
  HTTP/<Version> <Statuscode> <Statustext>
- Die wichtigsten Status-Codes sind:
  - **200**: Operation erfolgreich ausgeführt.
  - **404**: Die per URI angefragte Ressource konnte nicht gefunden werden.

Ein typischer Ablauf kann demzufolge wie folgt aussehen:

- Der Web-Browser sendet als Client einen Request:
  GET /test.htm HTTP/1.1 Host: localhost:8008
- Der Server antwortet mit der Response:

```
HTTP/1.1 200 OK
Date: Sat, 23 Mar 2019 15:12:48 GMT
Last-Modified: Sat, 23 Mar 2019 11:18:20 GMT
Content-Language: de
Content-Type: text/html; charset=utf-8

<… Inhalt der Datei test.htm …>
```

Wichtig: Hier wird ein doppelter Zeilenvorschub gesendet, den Sie beim Einlesen berücksichtigen müssen. Der Formatstring hierfür lautet **\r\n\r\n.**

3.1 Webserver für Textdateien (4 Punkte)

Der von Ihnen zu entwickelnde **mehrstufige** Web-Server soll folgende Anforderungen erfüllen:

1. Der Server wird unter Angabe eines Verzeichnisses, in dem HTML-Dokumente gespeichert sind, und eines Ports gestartet: **`httpserv <docroot> <port>.`** Bekanntlich wird üblicherweise der Port **`80`** verwendet, dessen Benutzung aber spezielle Privilegien erfordert. Verwenden Sie deshalb Port **`8080`** oder **`8008`**. Verwenden Sie wieder Chunks, um ressourcensparend zu arbeiten und auch mit sehr große Dateien zurecht kommen zu können.
2. Zunächst sollen reine **`html`**-Dokumente du die die Verarbeitung von **`GET`**-Requests unterstützt werden. Eine Beispieldatei **`funktioniert.html`** findet sich in **`Input_prakt3.zip`**. Die Datei, die im Request in der URI angegeben wurde, wird im Arbeitsverzeichnis gesucht, ausgelesen und an den aufrufenden Client zurückgegeben (Statuscode `200`). Wird die angegebene Datei nicht gefunden, wird der `404` zurückgegeben.
3. Sehen Sie Testausgaben zwecks Kontrolle der Verarbeitung der Client-Anfragen vor und achten Sie auf eine Fehlerbehandlung.
4. Testen Sie Ihren Server mit 2-3 Browsern und notieren Sie Unterschiede. Melden Sie sich, wenn Ihr Programm mit keinem oder nur mit einem Web-Browser läuft.

Hinweis: C bietet einige hilfreiche String-Verarbeitungsfunktionen.
`int sscanf(const char *str, const char *format, ...);` ermöglicht formatiertes Einlesen einer Zeichenkette. Mit **`sscanf (header, "GET %255s HTTP/", url)`** können z.B. die Parameter eines **`GET`**-Requests aus dem Header extrahiert werden. **`strstr`** ermöglicht das zeilenweise Durchsuchen des Textes nach einer Teilzeichenkette. z.B. zur Ermittlung des Dateinamens aus dem URI.

3.2 Erweiterung durch Abfangen von Fehlerfällen (4 Punkte)

Der Web-Server soll nun Grafiken ausliefern. Sie werden in HTML-Seiten z.B. durch **`<IMG SRC=text.png>`** eingebettet. Wenn ein Web-Browser auf ein **`<IMG SRC=>`**-Tag trifft, generiert er eine neue **`GET`**-Anfrage für die Bilddatei. Sorgen Sie dafür, dass die **`Binär`**-Datei korrekt übertragen wird. Testen Sie Ihren Server mit **`der_werwolf.html`**, die auf das eingebettete Bild **`C_Morgenstern.jpg`** zugreift.

3.3 Formularauswertung (2 Punkte)

Erweitern Sie den Mini-Web-Server so, dass er POST-Anfragen mit Parametern auswertet. Hierzu soll der Server zunächst eine Formularseite `form.html` ausliefern, die folgendermaßen aussieht:

```
<HTML> <BODY><center>
    <FORM ACTION="ServerHTTPPost" METHOD="post">
          Bitte die erste Zahl eingeben:     <br>
          <INPUT TYPE="text" NAME="zahl1">   <br>
          Bitte die zweite Zahl eingeben:    <br>
          <INPUT TYPE="text" NAME="zahl2">   <br>
          <INPUT TYPE="submit" VALUE="abschicken">
         </FORM></center>
</BODY> </HTML>
```

In den Eingabefeldern können Benutzer(innen) zwei Zahlen angeben, die mit dem POST Request zum Server geschickt und dort miteinander multipliziert werden. Ein Beispiel einer Anfrage lautet:

```
POST /index.htm:Search HTTP/1.1
Host: www.lbst.ecs.hs-osnabrueck.de
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

zahl1=3&zahl2=4
```

Wichtig: Auch hier wird ein doppelter Zeilenvorschub gesendet, den Sie beim Einlesen berücksichtigen müssen.

Die Parameter für `zahl1` und `zahl2` können Sie ignorieren. Die eingegebenen Werte müssen nicht überprüft werden. Erzeugen Sie eine Web-Seite, auf der das Ergebnis folgendermaßen dargestellt wird.

```
<HTML>
    <BODY>
        <center><h1> Ergebnis: 12</h1></center>
    </BODY>
</HTML>
```

Testen Sie Ihren Web-Server erneut.

Hinweis zur Fehlersuche:

Kompilieren Sie mit **`gcc`** und **`-g`** zur Erzeugung von Debug-Informationen. Speicherprobleme können mit **`valgrind`** gefunden werden**:** **`valgrind -v <ausführbare Datei>`**
Auf manchen Linux-Systemen muss der Aufruf mit **`/usr/bin/valgrind`** erfolgen**.**

Hinweise zu hilfreichen C-Funktionen:

Beschreiben Sie, warum in Client und Server die Dateigröße unwichtig ist und nicht ermittelt werden muss (und darf) und ob das Programm bei großen Dateien mehr Heap-Speicher als bei kleinen benötigt. In dieser Aufgabe sind folgende C-Funktionen zu verwenden:

- **`FILE *fopen (const char *filename, const char *mode);`**
  Öffnen einer Datei zum Lesen (Server) oder Schreiben (Client). Achten Sie auf den korrekten Mode, je nach Dateityp.
- **`int fclose(FILE *stream);`**
  Schließen einer Datei.
- **`size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`**
- **`size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE* stream);`**
  Lesen von und Schreiben auf eine Datei.

Details zu den Funktionen finden Sie in den man-Pages auf der Linux-Konsole. Versuchen Sie nicht, diese Funktionen selbst nachzubauen oder die Aufgabe mit anderen als den angegebenen Funktionen zu lösen. Für die C-Programmierung finden Sie im OSCA das Buch von S. Grägert : ‚*POSIXProgrammierung mit UNIX*' als PDF. Dieses beinhaltet die Dokumentation der Befehle zur Dateibehandlung, die für die Bearbeitung der Aufgabe notwendig sind.

Fehlersuche in C:
Kompilieren Sie stets mit dem **`gcc`** und der Option −g zur Erzeugung von Debug-Informationen.
Das korrekte Arbeiten der dynamischen Speicherverwaltung kann mit dem Befehl **`valgrind`** überprüft werden: **`valgrind -v <ausführbare Datei>`**
Auf manchen Linux-Systemen muss der Aufruf mit **`/usr/bin/valgrind`** erfolgen**.**