Aufgabenblatt 2

Realisierung eines einfachen Web-Servers

In dieser Aufgabe sollen Sie einen einfachen Web-Server auf Basis von TCP-Sockets implementieren. Dazu muss Ihr Programm eine Teilmenge des HTTP-Protokolls implementieren. Dieses wird im späteren Verlauf der Vorlesung noch detaillierter vorgestellt werden. Für die hier anstehende Übung seien die folgenden Angaben gemacht:

- HTTP baut als Protokoll der Anwendungsschicht auf TCP auf.
- HTTP unterscheidet die folgenden zwischen Client und Server ausgetauschten Nachrichten:
    - vom Client initiierte *Requests* (Anfragen) und
    - die *Responses* (Antworten) vom Server.
- Eine Nachricht besteht aus einem Nachrichten-Kopf (*Header*) und einem Nachrichten-Rumpf (*Body*).
- Durch entsprechende Attribute im Header können verschiedene *Request-Methoden* definiert werden. Bei den **`GET`**-Requests handelt es sich um die häufigste Zugriffsmethode auf einen Server. Durch einen solchen Request wird ein Datum, zumeist eine in Form einer sog. URI (*Uniform Resource Identifier*) eindeutig definierte Ressource, an den Server übertragen. Die Länge eines **`GET`**-Requests ist in der Regel 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 auf dem Server nicht gefunden werden.

Ein typischer Ablauf kann demzufolge wie folgt aussehen:

- Der Client sendet einen Request:
  ```
  GET /der_werwolf.html HTTP/1.1
  Host: localhost:8080
  ```
- Der Server antwortet mit Header und Body in der Response:
  ```
  HTTP/1.1 200 OK
  Date: Tue, 06 Apr 2013 15:12:48 GMT
  Last-Modified: Tue, 04 Apr 2013 11:18:20 GMT
  Content-Language: de
  ```

```
Content-Type: text/html; charset=iso-8859-1

<… Inhalt der Datei der_werwolf.html …>
```

Zwischen Header und Body muss ein doppelter Zeilenvorschub gesendet werden. Der Formatstring dafür lautet \r\n\r\n.

Der von Ihnen zu entwickelnde Web-Server soll die folgenden Anforderungen erfüllen:

1. Ihr Programm soll als **mehrstufiger Server** ausgelegt sein.
2. Der Server wird unter Angabe eines **Ports** und eines **Verzeichnisses**, in dem **`html`**-Dokumente gespeichert sind, gestartet:
   `HTTPserv <docroot> <port>`
3. Bekanntlich wird üblicherweise der Port **80** verwendet, dessen Benutzung aber spezielle Privilegien erfordert. Verwenden Sie deshalb Port **8080** oder **8008**.
4. Der Server muss nur die Verarbeitung von **`GET`**-Requests unterstützen:
   a. Die in dem mit dem Request übermittelten URI angegebene Datei wird in dem Arbeitsverzeichnis gesucht, ausgelesen und an den aufrufenden Client zurückgegeben (Statuscode **200**). Kann die in der URI angegebene Datei nicht gefunden werden, wird der Statuscode **404** zurückgegeben.
   b. Wird in der URI ein Verzeichnis (Unterverzeichnis im Arbeitsverzeichnis des Servers) referenziert, so wird das Verzeichnis in Form eines **`html`**-Dokumentes ausgegeben. Die in dem Verzeichnis vorhandenen **`html`**-Dateien und Unterverzeichnisse sind in dem Dokument verlinkt.
5. Sehen Sie die korrekte Behandlung von Dateitypen (z.B. Bilddateien im **`jpg`**-Format) vor. Dazu muss der *Content-Type* im Response-Header angepasst werden und Sie müssen die Dateien *binär* übertragen.
6. Im Dateibereich der Veranstaltung finden Sie ein Beispiel-Verzeichnis, welches durch Ihren Server abrufbar sein soll.
7. Sehen Sie Testausgaben zwecks Kontrolle der Verarbeitung der Client-Anfragen vor und achten Sie auf eine angemessene Fehlerbehandlung.
8. Testen Sie Ihren Server mit mehr als einem Browser und notieren Sie die Unterschiede bei der Verarbeitung von Anfragen.

Freiwillige Zusatzaufgabe:

Erweitern Sie Ihre Implementierung des Web-Servers so, dass er auch spezielle **`POST`**-Anfragen mit Parametern auswertet. Hierzu soll der Server zunächst eine Formularseite **`form.html`** ausliefern, die folgendermaßen aussieht:

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html">
        <title>Formular mit POST</title>
    </head>
    <body>
        <h2>Formular mit POST</h2>
        <form action="ServerHTTPPost" method="post">
            <table>
```

```
                <tr>
                    <td>Bitte die erste Zahl eingeben:</td>
                    <td><input type="text" name="zahl1"></td>
                </tr><tr>
                    <td>Bitte die zweite Zahl eingeben:</td>
                    <td><input type="text" name="zahl2"></td>
                </tr>
            </table><br>
            <input type="submit" value="Senden">
        </form>
    </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: localhost:8080
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

zahl1=3&zahl2=4
```

Auch hier doppelter Zeilenvorschub \r\n\r\n.

Liefern Sie eine Web-Seite zurück, auf der das Ergebnis der Multiplikation dargestellt wird:

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

Testen Sie die ergänzte Funktionalität.

Hinweise:

- Die Fehlersuche bei einem mehrstufigen Server kann aufwändig sein. Will man mit dem Debugger (anstelle von Testausgaben) arbeiten, sollte *zunächst ein iterativer Server* implementiert werden. Kompiliert werden muss dann mit **`-g`** Option. Funktioniert dieser problemlos, kann die Abarbeitung der Requests in einem Kindprozess ergänzt werden.

  Speicherprobleme können mit **`valgrind`** gefunden werden (**`valgrind -v <Programm>`**). Auf manchen Linux-Systemen muss der Aufruf mit **`/usr/bin/valgrind`** erfolgen

- Beim Abschicken einer URL in der Adressleiste im Browser können u.U. mehrere Requests ausgelöst werden. Alternativ können Sie einen korrekt formatierten Request auch über einen Telnet-Client verschicken.

- Bei der Erzeugung der Response sollten Sie mit Funktionen wie **`strcat()`**, **`sprintf()`**, **`strlen()`**,… nur zur Header-Generierung arbeiten. Soll z.B. eine Binär-Datei übertragen

werden, so lesen Sie diese **Byte-weise** per **`read()`** und schreiben die gelesenen Dateien per **`write()`** direkt auf das Socket.

- Neben den Ihnen aus der Vorlesung bekannten werden einige zusätzliche **C-Systemfunktionen** (Informationen dazu finden Sie im Online-Manual auf den Rechnern) benötigt:
  - **`FILE *fopen (const char *filename, const char *mode);`**
    Öffnen einer Datei zum Lesen (Server) oder Schreiben (Client).
  - **`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.
  - **`int lstat (const char* filename , struct stat* buf );`**
    Abfrage von Datei-Eigenschaften.
  - **`DIR* opendir (const char * name );`**
    Öffnen eines Verzeichnisses zum Lesen.
  - **`struct dirent* readdir (DIR *dir );`**
    Lesen der Einträge eines Verzeichnisses.
  - **`int closedir (DIR *dir );`**
    Schließen eines Verzeichnisses.
  - **`int sscanf (const char *str, const char *format, ...);`**
    Formatiertes Einlesen einer Zeichenkette. Mit dem Befehl **`sscanf (header, "GET %255s HTTP/", url)`** können z.B. die Parameter eines **`GET`**-Requests recht einfach aus dem Header extrahiert werden. Alternativ kann auch mit der Methode **`strtok()`** gearbeitet werden, wobei dies einen höheren Programmieraufwand erfordert.
  - Für die C-Programmierung finden Sie im OSCA das Buch von S. Grägert : ‚*POSIX-Programmierung mit UNIX*' als PDF. Dieses beinhaltet die Dokumentation der Befehle zur Dateibehandlung, die für die Bearbeitung der Aufgabe notwendig sind.

Für das Testat ist ein Protokoll bekannter Formatierung (siehe Blatt 1) inkl. der durchgeführten Tests vorzulegen.

Testierung: 26. / 27.4.2021