Aufgabenblatt 3[1]

Programmierung eines eigenen HTTP/1.1 Web-Servers

Ziele

- Weitere Analyse von Protokollabläufen mit Wireshark
- Einarbeitung in eine RFC-Spezifikation
- Entwicklung und Test eines einfachen Webservers

Ergebnis

- Maximal 3-seitiges Praktikumsprotokoll (deutsch oder englisch), das die angeforderten Teilaufgaben umfasst und in geeigneter Form darstellt. Eine Auflistung von Stichpunkten ist nicht ausreichend. Das Protokoll muss von einer dritten Person ohne Vorlage der Aufgabe verstanden werden können.
- Bearbeiten Sie das Aufgabenblatt bereits vor Beginn des Praktikumstermins vollständig. Halten Sie das Protokoll und Ihren Lösungsansatz zum Abnahmegespräch bereit. Es ist erforderlich, dass Sie Ihren Bildschirm während des Abnahmegesprächs teilen. Überprüfen Sie die *Screen-Share*-Funktionalität vor Praktikumsbeginn in ihrem Team.
- Bei nicht ausreichender Leistung wird vor dem nächsten Praktikumstermin eine weitere Nacharbeitung zugelassen. Eine Nichterfüllung im angesetzten Zeitrahmen führt zum Ausschluss.
- Das Protokoll inkl. des Lösungsansatzes ist innerhalb einer Woche nach dem Praktikumstermin abzugeben. Details zur Abgabe finden Sie im MS Teams Raum im Kanal Praktikum.

A Pflichtteil

> In dieser Aufgabe darf zur HTTP-Implementierung ausschließlich die low-level Berkeley Socket API genutzt werden.

A.1 HTTP Webserver

Nun soll ein einfacher Webserver in *C/C++*[2] entwickelt werden. Der Webserver soll als Parameter ein freigegebenes Verzeichnis haben, welches die Dateien der Webseite enthält (`DOCUMENT_ROOT_FOLDER`). Zudem soll als Parameter eine Logdatei angebbar sein, welche ein Log aller Zugriffe speichern soll. Listing 1 zeigt die Usage-Information.

```
1 Usage: http_server [OPTION...] DOCUMENT_ROOT_FOLDER
2 http_server -- A simple HTTP/1.1 web server.
3
4   -p, --port=PORT          Specify listening PORT
5   -l, --logfile=LOGFILE    Logfile to write to instead of stdout
6   -?, --help               Give this help list
7       --usage              Give a short usage message
8   -V, --version            Print program version
```

Listing 1: `http_server` Usage-Information

[1] basiert auf der Aufgabenstellung von Thomas Dreibholz
[2] oder *Rust*

Anforderungen

- Der Webserver muss nur HTTP `GET`-Requests bedienen können.
- Der Webserver muss auch nur einen Teil einer Ressource ausliefern können, wenn ein HTTP-Request dies fordert. Diese Anforderung muss wie oben (d.h. mit einfachen Bereichen wie `1024-` oder `1024-2047`) für Dateien unterstützt werden. Unterstützung beim Directory Listing ist nicht erforderlich.
- Der Webserver muss mehrere Verbindungen bzw. Clients gleichzeitig bedienen können.
- Ist das von einem Client angefragte Ziel eine Datei (z.B. *index.html*), so soll diese ausgeliefert werden. Der Content-Type soll anhand gängiger Dateiendungen ermittelt werden.
- Ist das von einem Client angefragte Ziel eine Verzeichnis, und existiert keine Datei *index.html* in diesem Verzeichnis, so soll ein Directory Listing zurückgeliefert werden. Diese HTML-Resource soll vom Server dynamisch generiert werden. Das Directory Listing soll Dateien, deren Länge und Datum/Uhrzeit der letzten Änderung z.B. als Tabelle darstellen. **Tipp**: `opendir()`, `closedir()`, `readdir()` und `stat()`. Existiert jedoch eine Datei *index.html* im Verzeichnis, soll diese zurückgeliefert werden.
- Man beachte, dass der Request vom Client aus dem Internet ($\rightarrow$ Angriffe!) kommen kann, d.h. der Webserver muss alle Eingaben auf Gültigkeit prüfen und ggf. die Verbindung abbrechen (z.B. Request zu lang, ungültige Anfrage, ungültige Content-Range, etc.). Im Fehlerfall soll der Webserver einen sinnvollen Fehler zurückgeben. Schauen Sie sich hierzu die HTTP-Statuscodes an.
- Der Webserver darf nur Dateien im freigegebenen Verzeichnis (und dessen Unterverzeichnissen) ausliefern!
- Zugriffe und Fehler sollen in der Logdatei mit Datum, Uhrzeit, IP-Adresse des Clients und angefragte Datei/Verzeichnis + Ergebnis bzw. Fehlermeldung festgehalten werden.
- Beim Empfang der Daten vom Client (z.B. mit `recv()`) ist darauf zu achten, dass TCP einen Bytestrom liefert. Unter Umständen übergibt der erste `recv()`-Aufruf nicht den vollständigen HTTP-GET-Request (siehe z.B. die Slow-Motion Option der entwickelten HTTP-Clientanwendung). Es muss also ggf. ein Teil des Requests zwischengespeichert werden, bevor der Request komplett ist.

A.2 Evaluation des Webservers

Entwickeln Sie Experimente, um Ihren Webserver auf Herz und Nieren zu überprüfen. Dokumentieren Sie die Experimente im Protokoll und führen Sie diese im Praktikum vor.

Ihr Webserver soll für die Experimente das *testweb* Verzeichnis freigeben. Das *testweb* Verzeichnis finden Sie unter `https://cloud.haw-hamburg.de/index.php/s/JDWMQRbEHGKXqbR`. Das Passwort lautet: `rnp$2021`.

Nutzen Sie für die Experimente den von Ihnen entwickelten HTTP-Clientanwendung und Werkzeuge wie `wget` und `curl`. Schauen Sie sich zudem noch Werkzeuge wie `diff` und `sha256sum` an. Für viele der Tests bietet es sich an, z.B. ein Skript zu schreiben, welches die Tests automatisiert durchführt. So lässt sich eine Testreihe leicht wiederholen, um auf Fehler nach Programmänderungen zu prüfen.

Beginnen Sie beispielsweise mit dem Download einer Textdatei. Laden Sie die Datei komplett und in Stücken herunter. Überprüfen Sie, ob Sie sie Datei wieder komplett zusammensetzen können. Entwickeln Sie insbesondere auch für die Fehlerfälle, die von Ihrem Webserver abgefangen werden sollen, Experimente.