Aufgabenblatt 2[1]

Programmierung einer eigenen HTTP/1.1 Clientanwendung

Ziele

- Weitere Analyse von Protokollabläufen mit Wireshark
- Einarbeitung in eine RFC-Spezifikation
- Entwicklung und Test einer HTTP/1.1 Clientanwendung

Ergebnis

- Maximal 3-seitiges (bezogen auf den Fließtext) 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 Einarbeitungsphase

Der RFC 2616[2] enthält die Spezifikation von HTTP/1.1. Machen Sie sich mit dem grundlegenden Protokollaufbau von HTTP vertraut und schauen Sie sich den Aufbau von *Request* und *Response* an. Sehen Sie sich im RFC 2616 zudem noch die wichtigsten HTTP Header-Felder an. Hierzu gehören u.a. `Content-Type`, `Range` und `Content-Range`.

Nutzen Sie ein Konsolenwerkzeug wie `wget` oder `curl`, um sich mit dem HTTP Protokoll vertraut zu machen. Fordern Sie beispielsweise nur einen Teil einer Ressource an und zeichnen Sie die Kommunikation in Wireshark auf.

Protokollieren Sie ihre Aufrufe und Beobachtungen. Nennen und erläutern Sie knapp die wichtigen HTTP-Header Felder.

[1] basiert auf der Aufgabenstellung von Thomas Dreibholz

[2] `https://www.rfc-editor.org/rfc/rfc2616.html`

A.2 HTTP Clientanwendung

In dieser Teilaufgabe soll eine einfache HTTP-Clientanwendung mit dem Namen `snatch` in *C/C++*[3] unter Linux entwickelt werden. Diese Anwendung soll später genutzt werden, um den eigenen HTTP-Server zu testen. Die mindestens zu unterstützenden Funktionen sind in Listing 1 dargestellt. Weitere Informationen werden im Abschnitt Anforderungen beschrieben.

```
Usage: snatch [OPTION...] URL
snatch -- A simple HTTP/1.1 client.

  -o, --output=FILE          Output to FILE instead of stdout
  -O, --remote-name          Output to FILE, named like the re-
                             mote document (extracted from URL),
                             instead of stdout
  -r, --range=RANGE          Retrieve a byte range (i.e. partial
                             document)
  -s, --slow=BYTES,TIMEOUT   Transmit request in chunks of BYTES
                             every N milliseconds (TIMEOUT) -
                             emulates a typed request.
  -v, --verbose              Produce verbode output to stderr
                             (request and response metadata)
  -?, --help                 Give this help list
      --usage                Give a short usage message
  -V, --version              Print program version
```

Listing 1: `snatch` help

Anforderungen

- Die Anwendung muss in der Lage sein eine Ressource von einem Webserver mittels HTTP-GET *Request* anzufordern.
- Der *Response*-Datenstrom (die angeforderten Ressource) ist auf `stdout` auszugeben. Durch den Schalter `-v` werden zudem die Metadaten des *Requests* und der *Response* auf `stderr` ausgegeben.
- Die Anwendung muss es erlauben nur einen Teil einer Ressource anzufordern und herunterzuladen. Hierzu ist der Schalter `-r` zu implementieren. Es ist ausreichend die folgenden Optionen zu unterstützen:
  - Anforderung ab einem Byte bis zum Ende der Ressource.
    Beispiel: `-r 1024-` → Anforderung von Byte 1024 bis zum Ende
  - Anforderung ab einem Byte bis zu einem anderen Byte.
    Beispiel: `-r 1024-2047` → Anforderung von Byte 1024 bis 2047

  Kompliziertere Optionen müssen nicht unterstützt werden!
- Die Anwendung stellt zudem eine Slow-Motion Option bereit. Hierzu ist der Schalter `-s` zu implementieren. Die Slow-Motion Option soll den HTTP-GET-*Request* in eine gegebene Anzahl von Bytes aufspalten und diese Blöcke getrennt mittels `send()` schicken, wobei nach jedem `send()`-Aufruf eine gegebene Zeitspanne gewartet wird. Die Option soll das manuelle Eintippen eines *Requests* (z.B. mittels `telnet` zum Server) emulieren. Die Angabe `-s 4,500` würde dann alle 500 ms bis zu vier Bytes des *Requests* schicken.
- Weitere zu unterstützende Optionen sind dem Listung 1 zu entnehmen.

[3] Wer möchte darf auch in *Rust* implementieren

Evaluation

Demonstrieren Sie die Funktion Ihrer HTTP-Clientanwendung `snatch` im Abnahmegespräch durch geeignete Aufrufe. Dabei sollten auch Grenzfälle und Ausnahmebehandlungen in den Experimenten abgedeckt werden.

In Listing 2 sind beispielhaft zwei Aufrufe aufgeführt. Entwickeln Sie weitere interessante Experimente.

```
1 ./snatch http://scimbe.de/_index.html
2 ./snatch http://localhost:8080/text/rfc793.txt
```

Listing 2: Ein Anfang ...

Wichtige Hinweise

- Jedes Teammitglied muss in der Bearbeitung der Aufgaben eine aktive Rolle einnehmen. Überlegen Sie sich wie sie ihre Ressourcen möglichst effektiv nutzen können und verteilen sie Zuständigkeiten. Es können durchaus einige Aufgabenteile unabhängig voneinander bearbeitet bzw. implementiert werden.

- Im Rahmen der Dokumentation sollten mindestens zwei UML-Diagramme sinnvoll eingesetzt werden. Abbildungen und Text sollen sich gegenseitig unterstützen. Ihr Protokoll dient als Diskussionsgrundlage im Abnahmegespräch.

- Beachten sie, dass es sich bei der angeforderten Ressource nicht ausschließlich um Textdateien handelt. Überlegen Sie sich daher wie sie sowohl mit Text- als auch Binärdateien umgehen können.

- Hilfreiche Module: Nutzen Sie beispielsweise ein Modul wie `argp`[4] zum Generieren des Hilfetextes und Parsen der Kommandozeilenparameter. Des Weiteren könnte die Verwendung von *Regulären Ausdrücken*[5] in Ihrer Lösung sinnvoll sein.

- Wie bereits aus dem Modul Betriebssysteme bekannt, sollten sie hinter sich aufräumen und die Speicherbereiche, die sie beispielsweise mit `malloc()` anfordern, auch wieder freigeben.

- Die Funktion `getaddrinfo()` kann genutzt werden, um die passenden Strukturen für die Aufrufe `bind()` und `connect()` vorzubereiten. Listing 3 enthält eine Liste wichtiger Header für die Socketprogrammierung.

```
1 #include <sys/socket.h> // core functions: socket(), ...
2 #include <netinet/in.h> // address and protocol families
3 #include <arpa/inet.h>  // htonl, ntohl, ...
4 #include <netdb.h>      // getaddrinfo
```

Listing 3: Header Dateien für die Socketprogrammierung

[4] `https://www.gnu.org/software/libc/manual/html_node/Argp.html`

[5] *regex.h*: `https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/regex.h.html` | *regex*: `https://en.cppreference.com/w/cpp/regex`