Praktikum

Rechnernetze und Verteilte Systeme

Block 3

Web Services

07. – 18.12.2020

Präsenzaufgaben

Die folgenden Aufgaben werden im Termin besprochen.

Aufgabe 1:

Angenommen Sie möchten ein Programm schreiben, das eine Funktion auf einem entfernten Server per Socket-API aufruft und dabei als Argument ein Paar aus ID und Namen übergibt. Das Programm speichert das Paar intern wie in Listing 1 dargestellt.

Listing 1 Werte-Paar

```
typedef struct ValueType {
    int          id;
    const char*  name;
} ValueType;
```

(a) Kann der dargestellte `struct ValueType` direkt an `send()` übergeben werden? Spielt die Prozessorarchitektur hierbei eine Rolle?

(b) Funktionsaufrufe auf entfernten Systemen werden oft als sog. Web Services über HTTP ausgeführt (siehe REST). Wie löst HTTP die Probleme aus a) ?

(c) Wie können komplexe Datenstrukturen übertragen werden, bspw. verkettete Listen oder Binärbäume?

Aufgabe 2:

Beantworten Sie im Kontext vom Hypertext Transfer Protocol (HTTP) folgende Fragen:

(a) Welche HTTP Methoden gibt es und was machen diese?

(b) Welche HTTP Methoden sind idempotent?

(c) Was sind persistente und nicht-persistente Verbindungen?

(d) HTTP ist ein zustandsloses Protokoll. Was bedeutet das? Mit welcher Technik können Server z.B. trotzdem den Warenkorb einer Nutzers speichern?

Praktische Aufgaben

Die praktischen Aufgaben sind in Kleingruppen von i.d.R. 3 Personen zu lösen. Die Ergebnisse besprechen Sie im zweiten Termin mit dem Tutor. Reichen Sie deshalb bitte den Quelltext bzw. Lösungen dieser Aufgaben bis Mittwoch der zweiten Woche des Blocks (16.12) 20:00 Uhr per ISIS ein. Aufgaben werden sowohl auf Plagiate als auch auf Richtigkeit hin automatisch getestet, halten Sie sich deshalb genau an das vorgegebene Abgabeformat.

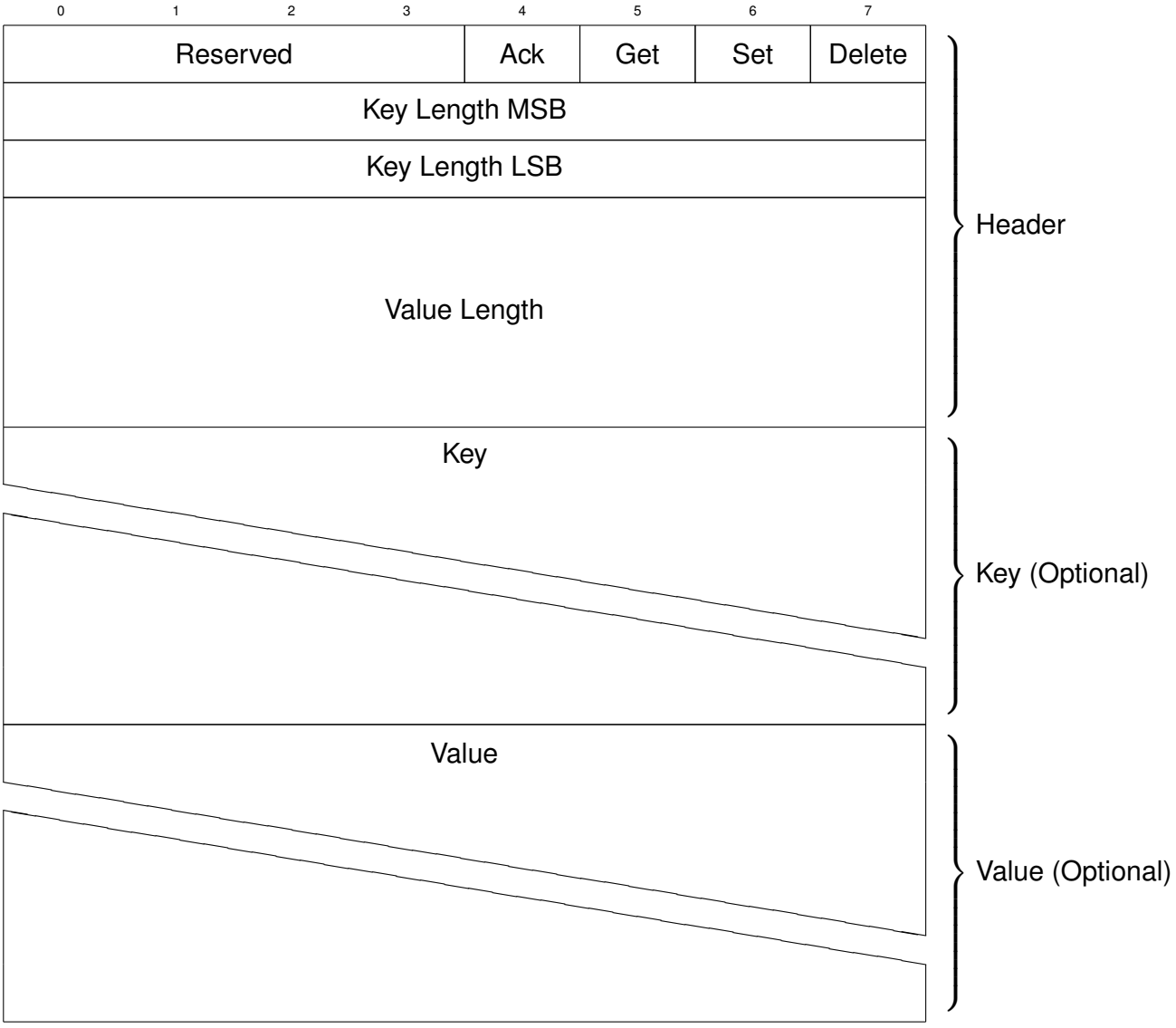
Sie haben in Block 2 einen Stream-Socket Client und Server bzw. einen Datagram-Socket Client geschrieben. Sie haben damit schon eine Art *Remote Procedure Call* (RPC) auf der Serverseite angeboten bzw. auf der Clientseite aufgerufen. Dabei war die Rückgabe eines Zitats der angebotene Service.

Wir wollen jetzt einen Schlüssel-Werte-Datenbank-Server implementieren, der intern eine Hash-Table zur Speicherung benutzt. In einer Hash-Table werden Tupel der Form `<key, value>` gespeichert [1]. Ihre Datenbank sollte die Befehle `set()` um einen Wert zu speichern oder zu aktualisieren, `get()` um einen Wert auszulesen und `delete()` um einen Wert zu löschen anbieten.

Client und Server verwenden für Aufrufe bzw. Rückgabewerte ein Protokoll, welches wir hier vorgeben, damit im Anschluss die Server und Clients aller Gruppen miteinander arbeiten können und ein automatisches Testen möglich ist. **Halten Sie sich deshalb genau an das Protokoll!** Das Protokoll basiert auf TCP, das heißt die unten

[1] https://de.wikipedia.org/wiki/Hashtabelle

angegebenen Nachrichten werden über einen zuverlässigen Bytestrom geschickt.



Das Protokoll ist aufgeteilt in einen festen Header, den Key mit variabler Länge und einen optionalen Value variabler Länge. Der Header enthält zunächst einige reservierte Bits. Diese sollen mit 0 gefüllt werden und könnten später z.B. für ein Versions-Feld benutzt werden. Danach folgen Bits für die Befehle bzw. die Server-Antwort. Der Client kann hier das Get, Set oder Delete Bit setzen. Ein gesetztes Bit bedeutet, dass es sich bei der Nachricht um diesen Anfragetyp handelt. Es soll dabei angenommen werden, dass immer nur ein Bit gesetzt ist. Der Server bestätigt alle Befehle bei erfolgreicher Durchführung mit einem Acknowledgment und setzt dafür das entsprechende Bit. Der

Server setzt auch die Bits für die Aktion, die erfolgreich durchgeführt wurde. Sie ermöglicht dem Client im Prinzip die Zuordnung des Acknowledgments zu der entsprechenden Anfrage.

Danach folgt die Länge des Keys als 16bit vorzeichenlose Ganzzahl und die Länge des Values als 32bit vorzeichenlose Ganzzahl. Alle Angaben beziehen sich falls nicht anders angegeben auf die Network-Byte-Order! Get und Delete-Anfragen enthalten nur einen Key, so dass die Länge des Values auf 0 gesetzt wird. Antworten auf Get-Anfragen enthalten Key und Value, die anderen Antworten enthalten weder Key noch Value. Nach diesem Header wird je nach Anfrage bzw. Antwort der Key gesendet, welcher variabel Lang sein kann. Darauf folgt optional der Value, der ebenfalls variabler Länge ist.

Halten Sie sich genau an die Vorgaben des Protokolls. Als Parameter soll ihr Server den Port übergeben bekommen. Ein Beispielaufruf sieht dann so aus:

```
$ ./server 4711
```

Achten sie bei der Implementierung besonders auf die folgenden Punkte:

(a) Wir wollen vereinfachend davon ausgehen, dass für jedes Anfrage-Antwort-Paar eine neue Verbindung aufgebaut wird. Der Server sollte die Verbindung nach der Antwort schließen.

(b) Der Hashtable-Teil kann beliebig in C implementiert werden, es ist hier auch erlaubt, eine externe Library zu verwenden, z.B. uthash [2]. Wenn Sie externe Libraries verwenden, müssen diese in Ihrer Abgabe enhalten sein!

(c) Wir werden als Keys *normalerweise* Strings verwenden, ihre Datenbank soll aber mit beliebigen Daten als Key bzw. Value arbeiten können. Keys und Values können dabei auch Null Bytes enthalten und sind nicht unbedingt Null-terminiert, deshalb sollten **keine String-Funktionen** benutzt werden. Achten Sie darauf, dass – falls sie uthash benutzen – die richtigen Makros für Ihren Datentyp verwenden. Auch die Längen von Key und Value soll variabel sein. Diese sollte dann entsprechend mitgespeichert werden.

[2] https://troydhanson.github.io/uthash/

(d) Sie können alle Keys und Values im Hauptspeicher ablegen.

Aufgabe 3:

Sie wollen nun zu ihrem Server den entsprechenden Client schreiben. Der Client soll als Kommandozeilenargument den DNS-Namen bzw. die IP-Adresse des Servers, die Methode (`SET`, `GET`, `DELETE`) und den Key übergeben bekommen. Geben Sie nur bei `GET` den zurückgegebenen Value ohne zusätzliche Zeichen aus und ansonsten nichts, damit wir automatisch testen können

Je nach Methode soll der Client von der Kommandozeile lesen und die Daten entsprechend als Value verwenden. Damit lässt sich dann bspw. eine Art Pseudo-Dateisystem nachbilden, indem Dateien mittels einer Pipe an ihren Client übergeben werden.

Einige Beispielaufrufe dazu:

```
$ ./client localhost 4711 SET /pics/cat.jpg < cat.jpg
$ ./client localhost 4711 GET /pics/cat.jpg > cat.jpg
$ ./client localhost 4711 DELETE /pics/cat.jpg

# < und > sind keine Parameter!
# Damit wird stdin/stdout umgeleitet
# Auch ist /pics/cat.jpg kein echter Pfad sondern nur der Key!
```

Achten sie bei der Implementierung besonders auf die folgenden Punkte:

(a) Sie können auf Client-Seite davon ausgehen, dass der Server die Verbindung nach der Antwort schließt.

(b) Sie können hier davon ausgehen, dass nur Strings als Kommandozeilenargumente übergeben werden. Der Value – in unserem Fall die Dateien – kann allerdings beliebige Bytes enthalten. Sie sollen in Ihrem Projekt CMake benutzen. Das Projekt sollte folgende Struktur aufweisen:

```
Block3
+--- CMakeLists.txt
+--- client.c
```

```
+--- server.c
```

Der folgende Aufruf soll im Unterverzeichnis `build` eine ausführbare Datei mit dem Namen `client` und eine mit dem Namen `server` erstellen:

```
$ mkdir build; cd build
$ cmake .. && make
```

Erstellen Sie aus Ihren **beiden** Lösungen ein tar.gz Archiv, z.B. so:

```
tar -czvf Block3.TXXGYY.tar.gz Block3
```

Laden Sie dieses Archiv bei ISIS bei der entsprechenden Abgabe hoch.

Vertiefungsaufgaben

Diese Aufgaben sind zu Ihrer eigenen Vertiefung in Hinblick auf die Klausurvorbereitung gedacht:

Aufgabe 4:

Gegeben sei ein RESTful Webservice, der Teilnehmer in einer Veranstaltung verwaltet. Der Service läuft auf dem Rechner mit dem DNS-Namen `api.tkn.tu-berlin.de}`. Folgendes Verhalten ist bereits vorgegeben und implementiert, dabei sind `<courseName>` und `<student>` jeweils Platzhalter für die jeweilige Veranstaltung bzw. den Teilnehmer:

Ressource	Methode	Verhalten
`/courses/<courseName>`	`GET`	Gibt Liste der Teilnehmer zurück
`/courses/<courseName>`	`DELETE`	Löscht die Veranstaltung
`/courses/<courseName>/<student>`	`GET`	Gibt bestimmten Teilnehmer zurück
`/courses/<courseName>/<student>`	`DELETE`	Löscht Teilnehmer

(a) Ist eine der Ressourcen eine Collection-URL und falls ja, geben sie die komplette URL an (inkl. verwendender Zugriffsmethode und Hostnamen)?

(b) Der Webservice soll von Ihnen nun um die Funktionalität erweitert werden, die Klausurnote eines Teilnehmers zu ändern. Welche HTTP-Methode muss dazu auf welche URL aufgerufen werden?

(c) Was wäre allgemein der Inhalt (Payload) der Anfrage aus Aufgabenteil b?

(d) Der Webservice soll von Ihnen nun um die Funktionalität erweitert werden, einen neuen Teilnehmer zur Veranstaltung hinzuzufügen, welcher vorher noch nicht angemeldet war. Welche HTTP-Methode muss dazu auf welche URL aufgerufen werden?

Aufgabe 5:

In dieser Aufgabe versuchen wir nachzuvollziehen wie ein Aufruf von YouTube abläuft und wie das verwendete CDN arbeitet. Rufen Sie dazu ein YouTube Video auf, während Sie den Netzwerktraffic mit Wireshark (siehe Blatt 1, Aufgabe 5) mitschneiden.

Hinweis: Durch gecachten Inhalt kann es sein, dass sie nicht die volle Anfrage sehen, falls Sie die Website bereits besucht haben. Um dies zu vermeiden starten Sie gegebenenfalls das System vor dem Test neu. Eure Teampartner erhalten unter Umständen andere Ergebnisse, vergleicht diese.

(a) Welche DNS Anfragen werden ausgelöst? Welcher der Hosts liefert vermutlich die eigentliche Website aus? Welcher Host stellt die Videodaten bereit?

(b) Wieso werden mehr Anfragen als nur `youtube.com` gestellt?

(c) Welche CDN Mechanismen werden verwendet?

(d) In der Vorlesung wurde besprochen, dass große Datenmengen (wie z.B. Videos) möglichst nah beim Nutzer gespeichert werden um Netzwerkkapazität zu sparen. Versuchen Sie herauszufinden, wo sich der Server befindet, der diese Daten ausliefert [3]. Überprüfen Sie ob dieser Standort plausibel ist, beispielsweise indem Sie die Latenz via `ping` testen.

[3] Bspw. über dieses Online-Tool: https://www.ip-tracker.org/locator/ip-lookup.php