Praktikum

Rechnernetze und Verteilte Systeme

Block 3

RPC

11.11. — 24.11.2019

1 Präsenzaufgaben

Die folgenden Aufgaben werden im Termin besprochen. Sie dienen auch der Vorbereitung der ISIS-Tests.

Aufgabe 1:

Nutzen sie das Beispiel des Abrufs von E-Mails durch Clients bei einem Server, um die folgenden Fragen für das Client-Server-Prinzip im Allgemeinen zu beantworten:

(a) Muss der betreffende Rechner (Client/Server) immer angeschaltet und ans Netz angeschlossen sein? Was hätte es für Konsequenzen, wenn beide Kommunikationspartner gleichwertig sind, also es keinen expliziten Server bzw. Client gibt?

(b) Welche Art von Adresse haben Client und Server? Was hätte es für Konsequenzen, wenn beide Kommunikationspartner gleichwertig sind, also es keinen expliziten Server bzw. Client gibt?

(c) Mit wem kommunizieren Clients in der Regel direkt? Mit wem Server?

Aufgabe 2:

Sie wollen einen Service per “Remote Procedure Call” (RPC) aufrufen und wissen aus der Vorlesung, dass Sie sich dazu zunächst eine Antwort auf folgende Fragen überlegen müssen:

(a) Was ist im Allgemeinen ein Service?

(b) Was bedeutet “idempotent” und wie vereinfacht eine solche idempotente Operation den RPC-Aufruf?

(c) Stellen Sie die Semantiken “at-most-once” und “at-least-once” gegenüber.

(d) Was ist der Unterschied zwischen asynchronem und synchonem RPC?

(e) Stellen Sie kurz die Übergabesemantiken *call-by-value* und *call-by-reference* des *klassischen* Prozeduraufrufs gegenüber.

(f) Sie möchten RPC mit den zuvor genannten Übergabesemantiken verwenden. Bei welcher Semantik sehen sie die größeren Probleme bezüglich der Umsetzbarkeit und wie lösen sie diese?

(g) Welche Formen der Transparenz gibt es? Welche Form der Transparenz steht für Sie beim RPC-Konzeptes im Vordergrund?

Aufgabe 3:

Ein Client nutzt synchrones RPC, um eine entfernte Prozedur mit Rückgabewert aufzurufen. Der Client braucht initial 5 Millisekunden, um seinen eigenen lokalen Client Stub mit den jeweiligen Parametern aufzurufen. Der Server braucht 10 Millisekunden, um seine lokale Prozedur aufzurufen und das Ergebnis zu erhalten. Die Verarbeitungsverzögerung (Processing Delay) für jede Sende- oder Empfangsoperation bei Client und Server sind jeweils 0,5 Millisekunden. Alle übrigen Verzögerungen (Queueing Delay (Warteschlangenverzögerung), Transmission Delay (Übertragungsverzögerung) und Propagation Delay (Ausbreitungsverzögerung)) addieren sich zwischen Client und Server in jeder Richtung auf jeweils 3 Millisekunden. Marshalling und Unmarshalling benötigen jeweils 0,5 Millisekunden.

(a) Wie lange dauert ein Aufruf vom lokalen Aufruf des Client Stubs bis zur Rückgabe des Ergebnisses an die aufrufende Prozedur?[1]

Aufgabe 4:

Sie möchten aus Ihrem Programm, das von einem 32-Bit-Prozessor mit Little-Endian-Darstellung ausgeführt wird, einen RPC aufrufen, der als Argument ein Paar aus ID und Namen verlangt. Das Programm speichert das Paar intern wie in Listing 1 dargestellt.

[1] Kontrollergebnis: 25ms

Listing 1: Werte-Paar

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

Listing 2: Doppelt-verkettete Liste

```
typedef struct ListNode {
    ValueType  value;
    ListNode*  prev;
    ListNode*  next;
} ListNode;

ListNode* myListHead;
ListNode* myListTail;
```

Listing 3: Binärer Baum

```
typedef struct TreeNode {
    ValueType  value;
    TreeNode*  left;
    TreeNode*  right;
} TreeNode;

TreeNode* myTreeRoot;
```

(a) Worauf müssen Sie beim (Un-)Marshalling dieses Werte-Paares achten?

(b) Ein weiterer RPC benötigt eine Liste dieser Werte-Paare. Die Liste wird intern gemäß Listing 2 gespeichert. Welche zwei zusätzlichen Probleme können hier auftreten?

(c) Treten alle Probleme aus b) auch bei einem binären Baum wie in Listing 3 auf?

2 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 Sonntag vor dem zweiten Termin (17.11.) 18:00 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.

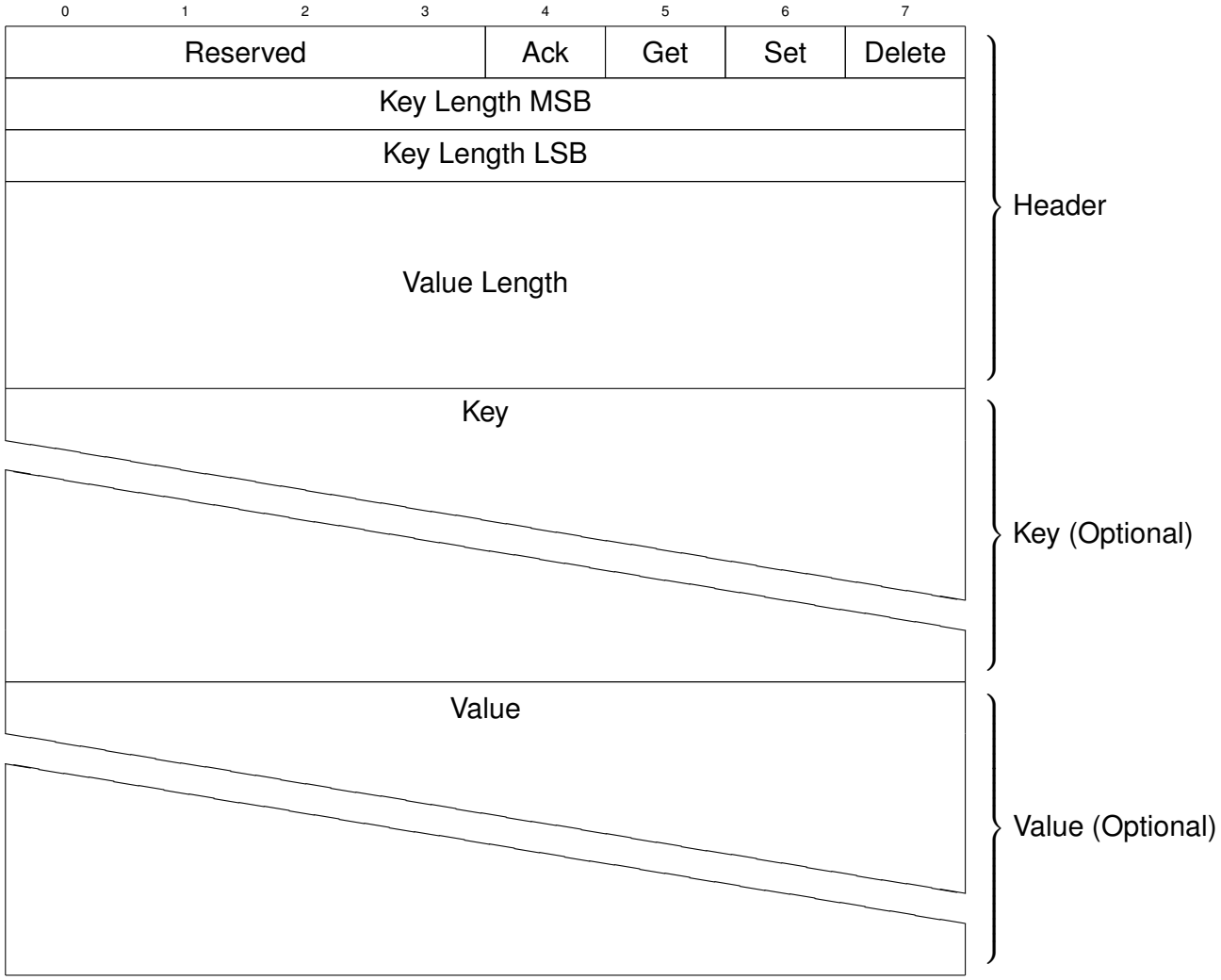
Aufgabe 5:

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-Datanbank-Server implementieren, der intern eine Hash-Table zur Speicherung benutzen. In einer Hash-Table werden Tupel der Form `<key, value>` gespeichert[2]. 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 angegebenen Nachrichten werden über einen zuverlässigen Bytestrom geschickt.

[2] `https://de.wikipedia.org/wiki/Hashtabelle`



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[3]. Wenn Sie externe Libraries verwenden, muessen 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.

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

Aufgabe 6:

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

[3] `https://troydhanson.github.io/uthash/`

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 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
```

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.

(c) 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** Lösung 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.

3 Vertiefungsaufgaben

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

Aufgabe 7:

Was bedeuten im Kontext von verteilten Systemen folgende Ausdrücke? Achten Sie auf eine möglichst genaue Definition!

(a) Autonomie

(b) Transparenz

(c) Skalierbarkeit

(d) Middleware

Aufgabe 8:

Sie haben in der Vorlesung gelernt, dass eine wesentliche Eigenschaft eines verteilten Systems das Verbergen der Verteilung ist. In diesem Zusammenhang haben Sie den Begriff der Transparenz kennengelernt. Welche Formen der Transparenz werden bei folgenden Beispielen (a und c) realisiert? Erläutern Sie diese bitte jeweils kurz!

(a) Beim NFS (Network Filesystem) wird ein auf einem Server befindliches Dateisystem beim Client in den lokalen Verzeichnisbaum eingehängt und erscheint wie ein lokales Verzeichnis mit den entsprechenden Dateien und Unterverzeichnissen des NFS Volumes. Programme greifen auf Dateien und Verzeichnisse des NFS Volumes mit denselben Systemaufrufen und Bibliotheksfunktionen zu, wie auf lokale Dateien/Verzeichnisse. Bezeichnet ein Name eine Datei auf dem NFS Volume, so werden Zugriffe auf die Datei automatisch durch das Betriebssystem mittels RPCs (Remote Procedure Calls) über das Netzwerk zum Server übertragen und ggf. der entsprechende Teil der Datei geändert (Schreiboperation) oder

an den Client übertragen (Leseoperation). Ist dies nicht erfolgreich, wird der RPC ggf. mehrfach wiederholt. Mehrere Clients können gleichzeitig lesend und schreibend auf verschiedene Dateien des Volumes oder nur lesend auf die selben Dateien zugreifen. Wird schreibend zugegriffen, so erscheint die Datei ggf. als hätte sie sich von selbst verändert, oder eine Datei kann explizit gegen gleichzeitigen Zugriff geschützt werden. Wird ein NFS Volume auf einen anderen Server verschoben, so kann nun dieses neue Volume unter dem bisher bekannten Verzeichnis eingehängt werden.

(b) Ist vollständige Failure Transparency realisierbar? Wie sieht das insbesondere in Hinblick auf das vorherige Beispiel aus?

(c) Ein Drucker wird über seinen Namen angesprochen. Zum Ausdrucken werden Druckjobs als Postscript-Dateien und unter Angabe des Druckernamens an den Druck-Server geschickt. Dieser konvertiert den Druckjob gegebenenfalls in die Sprache des Druckers, wenn der Drucker nicht Postscript-fähig ist. Mehrere gleichzeitige Druckjobs werden automatisch in eine Warteschlange eingereiht. Ist der Drucker überlastet (die Warteschlange zu lang) oder gar defekt, wird der Job automatisch an einen anderen Drucker weitergeleitet.

(d) Sind alle beim vorherigen Beispiel auftretenden Formen der Transparenz überhaupt sinnvoll? Begründen Sie Ihre Antwort.

Aufgabe 9:

Nennen Sie jeweils 3 Vor- und Nachteile der Client-Server-Architektur.

Aufgabe 10:

Die Zuordnung von Domainnamen zu IP-Adressen erfolgt im Internet mit Hilfe des Domain Name Systems (DNS). Beschreiben Sie anhand der folgende Fragen den Aufbau und die Funktionsweise von DNS:

(a) Wie sind die Verantwortlichkeiten im Namenssystem aufgeteilt?

(b) Wie funktioniert das rekursive bzw. iterative Auflösen von Namen mit DNS?

(c) Welche Funktion erfüllen die TTL-Einträge?

(d) Überprüfen Sie mittels des Kommandozeilen-Tools dig (Linux / OS X) bzw. nslookup (Windows), welche DNS-Server für die Domain tu-berlin.de sowie deren

Subdomains zuständig sind und welche TTL diese Einträge haben.