

Praktikum Rechnernetze und Verteilte Systeme

Block 3

— Verteilte Systeme und RPC —

Termin: 14.-16.11.2016 & 21.-23.11.2016

1 Theoretische Vorbereitungsaufgaben

Die folgenden Aufgaben sollen Ihnen helfen, sich auf den Vorbereitungstest vorzubereiten. Klären Sie bitte mögliche Fragen oder Unklarheiten unbedingt vor den ISIS-Testaten!

Aufgabe 1:

Sie kennen sich aus dem privaten und universitären Umfeld mit dem Konzept der “elektronischen Post” (E-Mail) aus. 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:

- 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?
- 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?
- Mit wem kommunizieren Clients in der Regel direkt? Mit wem Server?
- Wo sehen Sie Performanz-/Skalierbarkeitsprobleme?

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 synchronem 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.

- Wie lange dauert ein Aufruf vom lokalen Aufruf des Client Stubs bis zur Rückgabe des Ergebnisses an die aufrufende Prozedur?¹

2 Präsenzaufgaben

Die folgenden Aufgaben werden im Termin unter Anleitung des Tutors durchgeführt.

Aufgabe 4:

Tafelaufgabe - Wird im Termin vorgerechnet. Mitarbeit und damit Vorbereitung werden aber vorausgesetzt.

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.

- 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? (Tipp: Welche speziellen Eigenschaften besitzt ein Baum in der Informatik?)

Listing 1: Werte-Paar

```
1 typedef struct ValueType {  
2     int id;  
3     const char* name;  
4 } ValueType;
```

Listing 2: Doppelt-verkettete Liste

```
1 typedef struct ListNode {  
2     ValueType value;  
3     ListNode* prev;  
4     ListNode* next;  
5 } ListNode;  
6  
7 ListNode* myListHead;  
8 ListNode* myListTail;
```

¹Folien 34-36 von Unit 3 sollten hier helfen. Kontrollerggebnis: 25ms

Listing 3: Binärer Baum

```

1 typedef struct TreeNode {
2     ValueType value;
3     TreeNode* left;
4     TreeNode* right;
5 } TreeNode;
6
7 TreeNode* myTreeRoot;

```

3 Praktische Aufgaben

Die praktischen Aufgaben sind in Kleingruppen von i. d. R. 3 Personen zu lösen. Die Ergebnisse des ersten Termins führen Sie im zweiten Termin dem Tutor vor. Reichen Sie bitte den Quelltext bzw. Lösungen bis Sonntag vor dem zweiten Termin 23:55 Uhr per ISIS ein.

Im zweiten Termin werden vertiefende praktische Aufgaben gestellt, während der Tutor Lösungen des ersten Termins abnimmt. Reichen Sie bitte den Quelltext bzw. Lösungen dieser Aufgaben bis Sonntag vor dem nächsten Termin 23:55 Uhr per ISIS ein.

Es besteht in beiden Terminen grundsätzlich Anwesenheitspflicht.

3.1 Im ersten Termin zu lösen

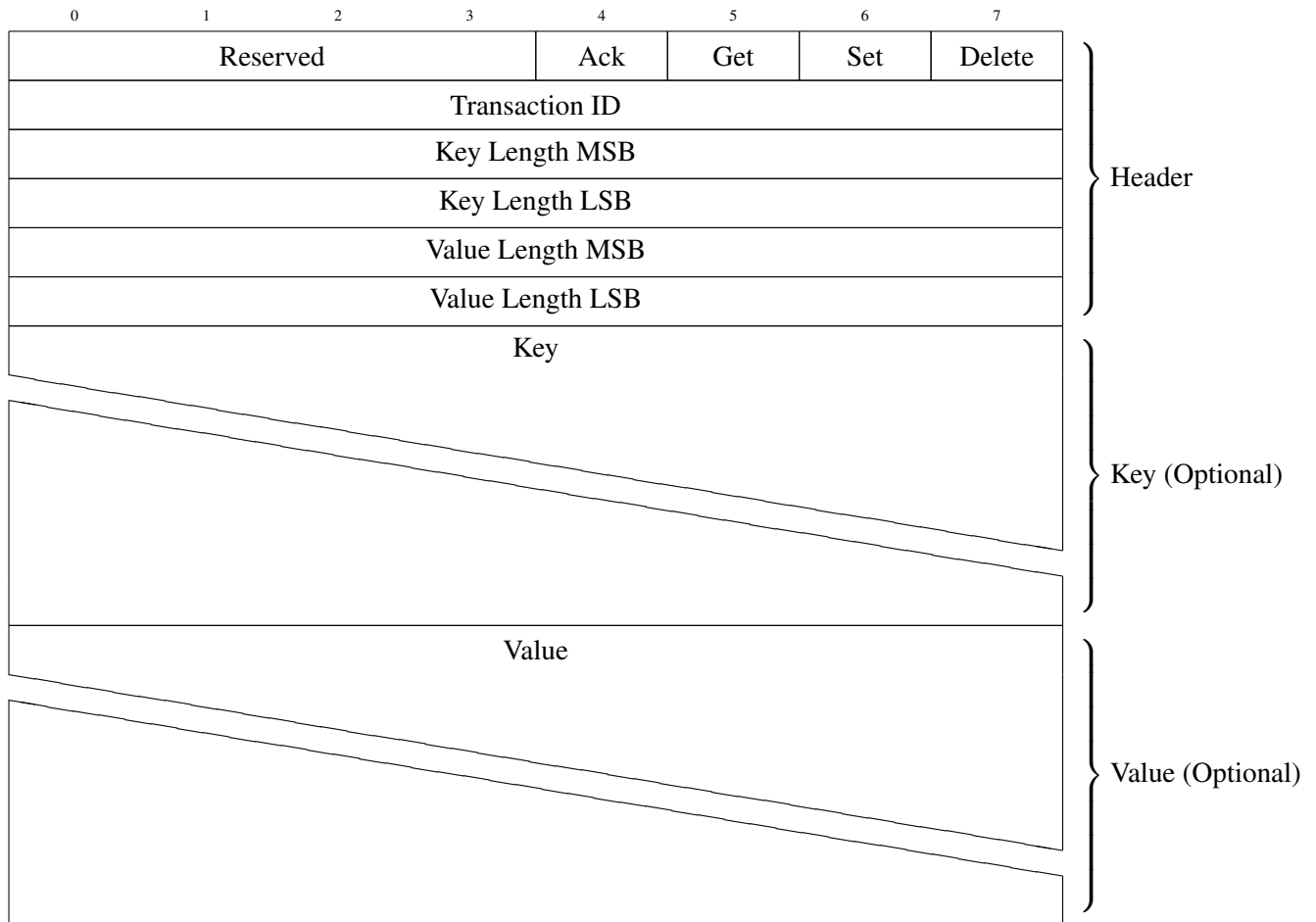
Aufgabe 5:

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 auf den nächsten Blättern verschiedene Micro-Services implementieren, die ein imaginärer Streaminganbieter² möglicherweise benötigt, um seinen Kunden die Plattform bereitzustellen. Die erste Aufgabe ist die Implementierung einer Film- bzw. Seriendatenbank. Wir wollen dazu der Einfachheit halber eine Hash-Table benutzen. In einer Hash-Table werden Tupel der Form `<key, value>` gespeichert. 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. Wir werden als Key in der nächsten Aufgabe den Filmtitel verwenden, ihre Datenbank soll aber mit beliebigen Daten als key bzw. value arbeiten können. Auch die Länge von key bzw. value soll variabel sein.

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. Das Protokoll basiert auf TCP, das heißt die unten angegebenen Nachrichten werden über einen zuverlässigen Byte-strom geschickt.

²<https://www.netflix.com/>



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 für ein Versions-Feld benutzt werden. Danach folgen Bits für die Befehle bzw. die Server-Antwort. Ein gesetztes Bit bedeutet hier, dass es sich bei der Nachricht um diesen Anfragetyp handelt. Der Client kann hier das Get, Set oder Delete Bit setzen. Es können dabei auch mehrere Befehle auf dem gleichen Datensatz in einer Nachricht abgesetzt werden. 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 Aktionen, die erfolgreich durchgeführt wurden. Darauf folgt eine Transaction-ID, die der Client beliebig setzen kann und in der Serverantwort unverändert enthalten ist. Sie ermöglicht dem Client die Zuordnung des Acknowledgments zu der entsprechenden Anfrage.

Danach folgt die Länge des Keys und die Länge des Values, jeweils als 16bit 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.

Die Hashtabelle kann beliebig in C implementiert werden. Wir haben als Tipp eine Hash-Funktion vorgegeben und würden das "chaining" mit verlinkten Listen als Kollisionsbehandlung empfehlen. Hier wird beim Einfügen bei einem bereits belegten Index das `<key, value>`-Paar in eine verlinkte Liste eingefügt. Beim Auslesen bzw. Suchen eines Eintrages muss am jeweiligen Index die verlinkte Liste durchgegangen werden, bis der Eintrag entweder gefunden wurde oder die Liste komplett durchgegangen wurde und der Eintrag nicht gefunden wurde.

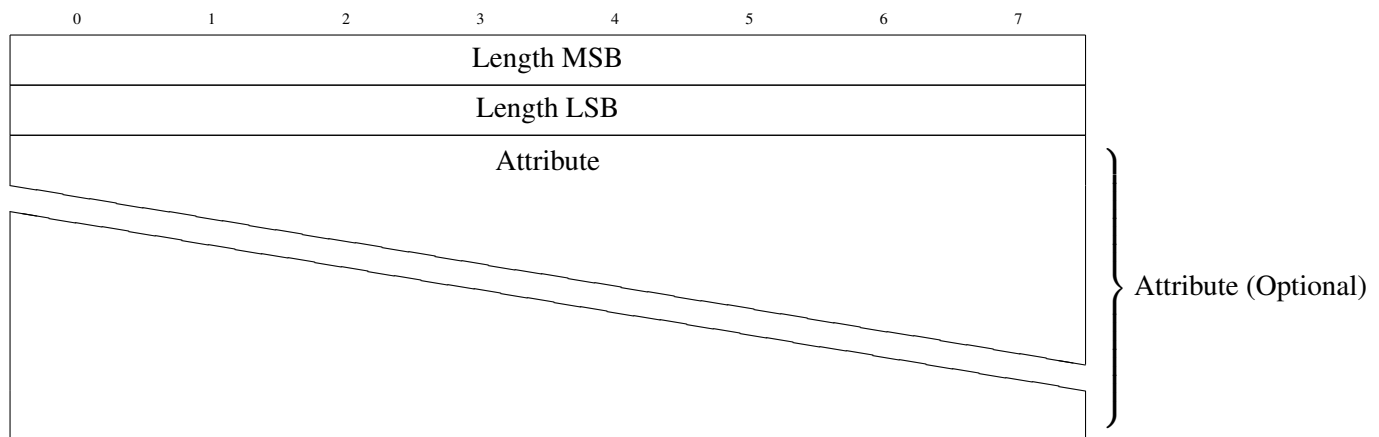
Halten Sie sich genau an die Vorgaben des Protokolls. Sie können Ihre Implementierung mit dem von uns bereitgestellten Client testen.

3.2 Im zweiten Termin zu lösen

Aufgabe 6:

Nehmen Sie als Ausgangspunkt Ihre Implementierung aus der letzten Aufgabe. Sie wollen nun zu ihrem Server den entsprechenden Client schreiben. **Der Client soll dabei beim Aufruf der Methoden Stubs benutzen, um die entfernte Ausführung des Prozeduraufrufs und damit einhergehende Übertragung von Argumenten und Ergebnis zu verstecken.** Gestalten Sie also die Schnittstellen bzw. Methodenaufrufe im Client so, als wäre die Funktionalität des entfernten Systems lokal vorhanden. Die nötige Vorbereitung des Sockets bzw. das Schließen darf weiterhin explizit passieren.

Als Key soll der Client den Filmtitel benutzen, der Value enthält mehrere Felder. Die Felder sind (in dieser Reihenfolge): Titel, Originaltitel, Herstellungsjahr, Länge, Regie, DarstellerInnen. Die Einträge stehen im Value direkt hintereinander und haben jeweils die folgende Struktur. Ist ein Eintrag nicht vorhanden, ist die Länge 0.



Der Client soll als Kommandozeilenargument den DNS-Namen bzw. die IP-Adresse des Servers und optional eine Datei übergeben bekommen, die er Zeile für Zeile ausliest und die Daten entsprechend in die Datenbank schreibt. Eine Beispiel-Datei findet sich auf der ISIS-Seite. Jede Zeile enthält einen Film-Datensatz. Einzelne Felder des Datensatzes sind durch ein Semikolon (;) voneinander getrennt. Schreiben Sie Ihren Client so, dass er 25 (z.B. zufällige) <key, value>-Paare hintereinander speichert, ausliest, löscht und abschließend nochmals (vergeblich) ausliest. Machen Sie durch Debug-Ausgaben klar, welchen Rückgabewert die entfernten Aufrufe jeweils gehabt haben.

Sie wollen darüber hinaus in einem zweiten Schritt garantieren, dass der Server die Anfrage auch tatsächlich mindestens einmal ausführt bzw. der Client nicht endlos auf eine Antwort wartet, wenn ein Fehler auftritt. Implementieren Sie deswegen eine at-least-once Semantik mit einem Timeout von 2 Sekunden. **Beantworten Sie zusätzlich die theoretische Frage, ob die at-least-once Semantik für eine Hash-Table Sinn macht.**

4 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!

- Autonomie
- Transparenz
- Skalierbarkeit
- 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!

- 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.
- Ist vollständige Failure Transparency realisierbar? Wie sieht das insbesondere in Hinblick auf das vorherige Beispiel aus?
- 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.
- 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:

- Wie sind die Verantwortlichkeiten im Namenssystem aufgeteilt?
- Wie funktioniert das rekursive bzw. iterative Auflösen von Namen mit DNS?
- Welche Funktion erfüllen die TTL-Einträge?