TKN Telecommunication Networks Group

Praktikum
Rechnernetze und Verteilte Systeme

Block 3

— Verteilte Systeme und RPC —

Termin: 10.-12.11.2014 & 17.-19.11.2014

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 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.

- 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

```
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;
```

[1] Folien 25 und 26 von Unit 3 sollten hier helfen. Kontrollergebnis: 25ms

Listing 3: Binärer Baum

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

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 des weiteren den Quelltext bzw. Lösungen bis Sonntag vor dem 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, bei dem der Client dem Server zwei natürliche Zahlen schickt. Dieser berechnet dann den größten gemeinsamen Teiler (GGT) und gibt das Ergebnis über die Standardausgabe aus. Dabei wurden die natürlichen Zahlen mit einer Funktion `packData` bzw. `unpackData` für das Senden vorbereitet.

Sie haben damit schon eine Art "Remote Procedure Call" (RPC) auf der Serverseite angeboten bzw. auf der Clientseite aufgerufen. Dabei war das Ausrechnen des GGTs der angebotene Service. Die Funktion `packData` bzw. `unpackData` waren das (Un-)marchalling der Daten. Sie sollen in dieser Aufgabe diesen RPC-Mechanismus zunächst wie folgt zu einem RPC auf Basis von Datagram-Sockets, welcher als Übergabesemantik "call-by-value" verwendet, erweitern:

a) Schreiben Sie Ihren Server aus Block 2 so um, dass er Datagram-Sockets statt Stream-Sockets nutzt.

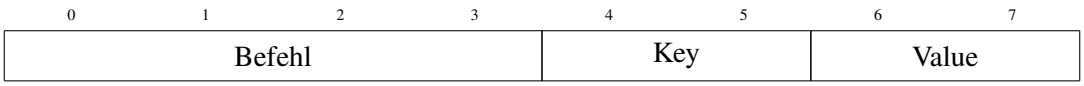
b) Schreiben Sie Ihren Client und Server so um, dass der Server den GGT nicht auf der Konsole ausgibt, sondern zurücksendet und der Client auf eine Antwort des Servers wartet und diese ausgibt. Dabei können Sie wieder die Funktion `packData` benutzen und die zweite Zahl mit Null-Bytes auffüllen. *Tipp: Kapitel 5.8 aus dem „Beej's Guide to Network Programming Using Internet Sockets"[2] könnte hilfreich sein!*

Aufgabe 6:
Der Server soll in dieser Aufgabe nun nicht den GGT zweier Zahlen berechnen, sondern das Speichern und Auslesen von Werten anbieten. Wir wollen dazu eine Hash-Table benutzen. In einer Hash-Table werden Tupel der Form `<key, value>` gespeichert. Ihe Hashtabelle 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.

[2] `http://beej.us/guide/bgnet/`

Der Client soll dabei beim Aufruf der oben genannten 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.

Client und Server erkennen Aufrufe bzw. Rückgabewerte für eine bestimmte Methode an einem String-Präfix. Dabei sollen die Präfixe von Nachrichten des Clients jeweils die Strings "SET", "GET" und "DEL" sein. Die Antworten des Servers sollen mit "VAL" das jeweilige Tupel zurückgeben bzw. mit "NOF" auf einen nicht gefundenen Eintrag und mit "OK!" für Erfolg oder "ERR" für Misserfolg auf die Anfragen zu "SET" und "DEL" antworten. Die Strings sollen dabei jeweils Null-terminiert sein, womit sich eine Größe von 4 Bytes ergibt. Der Einfachheit halber sollen Key und Value entweder wie mit den Methoden `packData` bzw. `unpackData` für die Übertragung vorbereitet werden, oder falls die Werte nicht gebraucht werden ebenfalls mit Null aufgefüllt werden. Das Nachrichtenformat ist also ähnlich wie in der letzten Aufgabe mit einem zusätzlichen String als Präfix (Angaben in Bytes):



Eine Hash-Table funktioniert dabei wie folgt. Eine "Hash-Funktion" wird auf die keys angewendet. Diese Funktion bildet die keys auf eine positive Ganzzahl mit einer oberen Grenze ab. Die keys werden dann in einem Array aus sogenannten Buckets am berechneten Index eingefügt, wobei die Größe des Arrays durch das Maximum der Ausgabe der Hash-Funktion gegeben ist. Beim lesen wird die Hash-Funktion auf den betreffenden key angewendet, um den entsprechenden Bucket mit dem `<key, value>`-Paar zu finden.

Im besten Fall verteilt die Hash-Funktion dabei die Keys möglichst zufällig und gleichmäßig auf die einzelnen Buckets. Dabei passiert es in der Regel, dass mehrere keys dem selben Bucket zugeordnet werden. Dieses Problem nennt man Hash-Kollision und muss von der Hash-Tabelle behandelt werden.

Ein einfacher Weg mit einer solchen Kollision umzugehen ist das "chaining" mit verlinkten Listen. 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.

a) Schreiben Sie Ihren vorhandenen Datagram-Socket Client und Server so um, dass ein RPC mit entsprechenden Stubs aufgerufen werden kann. Konzentrieren Sie sich zuerst auf den Austausch der Nachrichten und implementieren Sie noch keine Funktionalität.

b) 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.

c) Implementieren Sie anschließend auf der Server-Seite eine einfache Hash-Tabelle mit der "Hash-Funktion" nur ein Byte des Keys zu betrachten (entspricht 256 Buckets) und der Kollisionsvermiedung mittels "chaining". *Tipp: Die Hash-Tabelle muss dabei nur aus einem Array der festen Größe 256 bestehen. Als Elemente des Array bieten sich Zeiger auf structs mit Werten für key, value, next an.*

3.2 Im zweiten Termin zu lösen

Aufgabe 7:
Nehmen Sie als Ausgangspunkt Ihre Implementierung aus der letzten Aufgabe. Sie wollen nun 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 die Antwort verloren geht. Implementieren Sie deswegen eine at-least-once Semantik mit einem Timeout von 2 Sekunden. Einige Tipps:

- Wenn in einer gewissen Zeit keine Antwort vom Server erhalten wurde, müssen Sie die Anfrage erneut schicken.
- Am besten nutzen Sie dafür die Funktion `select()`, die im Zusatzmaterial zu Block 2 und in Kapitel 7.2 des „Beej's Guide to Network Programming Using Internet Sockets“[3] beschrieben ist, um den Deskriptor für 2 Sekunden auf neue Datagramme zu überwachen.

Aufgabe 8:
Im vorherigen Block haben Sie Ihren Stream-Socket Client so umschreiben, dass er Daten von HTTP-Servern abrufen kann. Wir wollen diesen nun Nutzen, um einen Wetter-Service über HTTP aufzurufen. Nutzen Sie Ihren Client und die Yahoo Weather API[4], um die Temperatur in Berlin auf der Konsole auszugeben. Die URL, die Sie dazu abrufen müssen, lautet `http://weather.yahooapis.com/forecastrss?w=638242&u=c`[5].

Antworten Sie zusätzlich auf die folgenden Fragen mit je einem Satz:

a) Warum meinen Sie nutzt dieser Webservice HTTP als Transport?

b) Sehen Sie sich die Antwort des Servers an. Würden Sie sagen, dass dieser Webservice “cachable” ist?

c) Erfüllt der Service die Zustandslosigkeit?

4 Vertiefungsaufgaben

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

Aufgabe 9:
Was bedeuten im Kontext von verteilten Systemen folgende Ausdrücke? Achten Sie auf eine möglichst genaue Definition!

- Autonomie
- Transparenz
- Skalierbarkeit
- Middleware

[3] `http://beej.us/guide/bgnet/`
[4] `https://developer.yahoo.com/weather/`
[5] Ein wichtiger Teil der Aufgabe besteht darin, nur die Temperatur auszugeben. Sie sollten sich dazu mit den String-Funktionen in C vertraut machen, z.B. unter `http://openbook.galileocomputing.de/c_von_a_bis_z/011_c_arrays_013.htm`

Aufgabe 10:
Sie haben in der Vorlesung gelernt, dass eine wesentliche Eigenschaft eines verteilten Systems das Verbergen der Verteiltheit 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 11:
Nennen Sie jeweils 3 Vor- und Nachteile der Client-Server-Architektur.

Aufgabe 12:
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?