# Verteilte Systeme (I6, Bc)

## Blatt 03: RPC-Programmierung mit SUN-RPC

**Sommersemester 2017**

Bearbeitung im Praktikum ab 18.05.2017

***Hinweise*:**

- Gegebenenfalls müssen Sie vor Start Ihres RPC-Servers den Binder mit `rpcbind` starten. Auf manchen Systemen benötigen Sie dafür **Root**-Rechte.
- Ist `rpcbind` noch nicht auf der Referenzplattform installiert, so können Sie die Installation mit: `sudo apt-get install rpcbind` durchführen.
- Zum Testen, ob der Binder (auch Portmapper genannt) auf Ihrem System läuft, in der Konsole `rpcinfo` oder `rpcinfo -p` (siehe auch man pages) ausführen.
- Für den Start Ihres RPC-Servers benötigen Sie auf manchen Systemen **Root**-Rechte, damit Ihr RPC-Server das Recht hat, sich beim Binder zu registrieren.
- Haben Sie Ihren RPC-Server gestartet, dann kann auf der Konsole mit `rpcinfo -p` (siehe auch man pages) nachgesehen werden, ob der RPC-Server wirklich läuft und seine Interfaces registriert sind. Mit `rpcinfo -d <prognumber> <version>` kann die Registrierung eines RPC-Servers im Binder gelöscht werden.

## Aufgabe 1: RPC-Server und RPC-Client – Basic

### 1.1 RPC-Server

Schreiben Sie einen RPC-Server (RPC-Server-Basic), der die folgende Funktionalität zur Verfügung stellt:

1. Funktion `vs_square()`,
   die einen long-Wert als Eingabe erhält, den Wert mit sich selbst multipliziert und das Ergebnis als Returnwert zurückgibt.
2. Funktion `vs_add()`,
   die zwei long-Werte addiert und die Summe als Returnwert zurückgibt.
3. Funktion `vs_concat()`,
   die zwei Zeichenketten als Eingabeparameter nimmt und die Konkatenation der beiden Zeichenketten als Returnwert zurückliefert.
4. Funktion `vs_split()`,
   die eine Zeichenkette als Eingabeparameter nimmt, die Zeichenkette beim ersten White-Space-Zeichen (Leerzeichen, Tabulator, Carriage Return, Line Feed) aufspaltet und die beiden Teile als Ergebnis liefert.VS_
5. Funktion `vs_increment()`,
   die einen long-Wert incrementiert, aber als Returntyp void hat.

6. Außerdem soll der RPC-Server eine Funktion `vs_shutdown()` bieten, über die der RPC-Server beendet werden kann. Beim Beenden soll der Server auch seine Registrierung beim Binder löschen und belegte Ressourcen freigeben.

Machen Sie durch geeignete Ausgaben auf der serverseitigen Konsole kenntlich, wenn eine dieser Funktionen beim RPC-Server aufgerufen wurde.

### 1.2 RPC-Client

Schreiben Sie einen RPC-Client, um Ihren RPC-Server-Basic zu testen.

- Jede der oben genannten Interface-Funktionen (außer `vs_shutdown()`) soll dabei mindestens zweimal, und zwar mit verschiedenen Eingabewerten aufgerufen werden.
- Statten Sie Ihren Client mit einer Aufrufoption aus, mit der es möglich ist, den Server zu beenden.
- Das Ziel der RPC-Aufrufe (IP-Adresse/Rechnername) soll bei Ihrem Client über Kommandozeilenargumente konfigurierbar sein.

**Zusätzliche Anforderungen:**

- Nutzen Sie RPC über TCP.
- Geben Sie in Ihren Programmen jeden allokierten Speicher wieder frei, insbesondere auch den Speicher, den das RPC-System allokiert und an Ihren Client/Server weitergibt.

***Hinweise*:**

1. Realisieren Sie die Funktionen des RPC-Servers Schritt für Schritt, beginnend mit der Ersten.
2. Überlegen Sie sich zuerst die Funktionsprototypen der zu realisierenden Funktionen. Teilweise sind diese ja in der Aufgabenstellung schon beschrieben.
3. Leiten Sie dann daraus die Schnittstellenbeschreibungsdatei (.x-Datei) mit den Interface-Funktionen ab.
4. Erzeugen Sie nun mit dem Werkzeug `rpcgen` die Header-Dateien und die Stubs für Ihren RPC-Server und RPC-Client. Mit `rpcgen -a` können Sie sich einen Beispielserver und einen Beispielclient erzeugen lassen.
5. Schauen Sie sich die generierten Dateien an und schlussfolgern Sie daraus, welche Funktion Sie auf der Serverseite implementieren müssen.
6. Nutzen Sie die Beispiele aus dem Skript für Ihre Implementierung und/oder suchen Sie nach Samples im Internet.
7. Prüfen Sie bei jedem Aufruf einer Interface-Funktion oder einer Funktion des RPC-Laufzeitsystems auf der Clientseite den Returnwert auf Fehler ab. Nutzen Sie die Funktion `clnt_perror()`, um den Fehlertext auszugeben.
8. Prüfen Sie, ob `clnt_create()` erfolgreich war. Evtl. aufgetretene Fehler mit `clnt_pcreateerror()` ausgeben.
9. Am Ende Ihres RPC-Client sollten Sie mit `clnt_destroy()` die vom RPC-Client allokierte RPC-Verbindung wieder freigeben.
10. Nutzen Sie, wenn geeignet/benötigt `xdr_free()`, um auf der Serverseite Speicher wieder freizugeben.


 

11. Nutzen Sie, wenn geeignet/benötigt `clnt_freeres()`, um auf der Clientseite Speicher wieder freizugeben.
12. Nutzen Sie `strdup()`, um Zeichenketten zu duplizieren.
13. Nutzen Sie `isspace()`, um die Trennstelle in `vs_split()` zu berechnen.
14. Verwenden Sie zur Realisierung der Funktion `vs_increment()` nicht länger als 15 Minuten. Wenn Sie es innerhalb dieser Zeit nicht schaffen, lassen Sie die Implementierung dieser Funktion weg.

## Aufgabe 2: RPC-Server und RPC-Client – BasicPlus

Schreiben Sie einen RPC-Server (RPC-Server-BasicPlus), der folgende Funktionalität bietet:

1. Funktion `vs_quad()`,
   die einen int-Wert als Eingabe hat und als Ergebnis ein Feld mit den Quadratzahlen von 1 bis zum Wert der Eingabe zurück gibt. Ist z.B. der Eingabewert gleich `3`, dann ist das Ergebnis das Feld `{1,4,9}`.
2. Funktion `vs_twice()`,
   die eine Zeichenkette `name` als Eingabe hat und als Ergebnis den Wert von `name` als ein Element und `name` mit sich selbst konkatentiert als ein anderes Element zurück liefert. Ist z.B. der Eingabewert gleich `"ab12"`, dann besteht das Ergebnis aus den beiden Elementen `"ab12"` und `"ab12ab12"`.
3. Funktion `vs_readdir()`,
   die eine Zeichenkette `dirname` als Eingabe hat und als Ergebnis den Inhalt des Verzeichnisses `dirname` zurückgibt. D.h. das Ergebnis besteht aus einer verketteten Liste von Datei- und Verzeichnisnamen bzw. aus einer Fehlerkennung, wenn das Verzeichnis `dirname` nicht existiert oder nicht gelesen werden kann.
4. Außerdem soll der RPC-Server eine Funktion `vs_shutdown()` bieten, über die der RPC-Server beendet werden kann. Beim Beenden soll der Server auch seine Registrierung beim Binder löschen und belegte Ressourcen freigeben.
5. Setzen Sie den Time-out für einen RPC-Aufruf auf zwei Minuten (Voreinstellung ist 25 Sekunden).

Schreiben Sie zum Testen Ihres neuen RPC-Servers-BasicPlus auch einen RPC-Client. Dieser RPC-Client soll jede Funktion des Interfaces mindestens einmal aufrufen.

***Hinweise*:**

1. Benutzen Sie Felder um das Ergebnis von `vs_quad()` zurückzugeben.
2. Mit `opendir()` und `readdir()` lassen sich Verzeichnisinhalte auslesen.
3. Nutzen Sie eine `union`, um das Ergebnis von `vs_readdir()` zurückzugeben.
4. Mit `clnt_control()` lässt sich der Wert für den Time-out setzen.
5. Übergeben Sie die Aufrufparameter für `vs_quad()` und `vs_readdir()` dem Client als Kommandozeilenparameter.


 

## *Optionale* Aufgabe 3: Mehrere Interfaces in einem RPC-Server

Fassen Sie die RPC-Server aus Aufgabe 1 und Aufgabe 2 zusammen, indem Sie einen neuen **RPC-Server-BasicAndPlus** schreiben, der zwei Interfaces hat. Ein erstes Interface soll die Funktionen enthalten, welche im Interface des RPC-Servers-Basic von Aufgabe 1 vorhanden sind, und ein zweites Interface soll die Funktionen besitzen, welche im Interface des RPC-Servers-BasicPlus von Aufgabe 2 existieren.

Schreiben Sie zum Testen Ihres neuen RPC-Servers-BasicAndPlus auch einen RPC-Client. Dieser RPC-Client soll jede Funktion der Interfaces mindestens einmal aufrufen.

***Hinweise*:**

1. Zur Lösung dieser Aufgabe müssen Sie sich Gedanken über die Bedeutung der Begriffe Interface, Programm und Version im Kontext einer RPC-Schnittstellendefinition machen.
2. Die Interfaces aus Aufgabe 1 und Aufgabe 2 sind generell verschieden. Deshalb sollten Sie für jedes Interface ein Element `program` in der Schnittstellendatei verwenden.
3. Wenn Sie nach Hinweis 2 vorgehen, werden sie einen Namenskonflikt erhalten. Lösen Sie diesen, indem Sie die Funktion, die diesen Konflikt verursachen, herauslösen und in ein eigenes Interface stecken.

## *Optionale* Aufgabe 4: Konkurrenter Zugriff auf einen RPC-Server

Um den konkurrenten Zugriff auf einen RPC-Server zu testen, implementieren Sie einen neuen Server (**RPC-Server-Simple**) mit folgendem Interface:

1. Funktion `vs_add()`,
   die zwei long-Werte addiert und die Summe als Returnwert zurückgibt. Außerdem schläft (Aufruf der Funktion `sleep()`) der RPC-Server in dieser Servicefunktion d Sekunden, bevor er die Berechung beginnt. Der Wert d ergibt sich aus dem ersten Operanden der Addition. Ist d dadurch >32, so wird d auf 32 gesetzt. Ist d<0, dann wird kein Schlafen ausgeführt.
2. Funktion `vs_concat()`,
   die zwei Zeichenketten als Eingabeparameter nimmt und die Konkatenation der beiden Zeichenketten als Returnwert zurückliefert. Außerdem schläft (Aufruf der Funktion `sleep()`) der RPC-Server in dieser Servicefunktion d Sekunden, bevor er die Berechung beginnt. Der Wert d ergibt sich aus dem ASCII-Wert des ersten Buchstabens des ersten Operanden der Konkatenation. Der sich zunächst so ergebende Wert wird modulo 32 genommen und dann der Wert 1 abgezogen. Ist d<0, dann wird kein Schlafen ausgeführt.
3. Funktion `vs_sleep()`,
   die einen int-Wert als Dauer für das Schlafen der Serverfunktion in Sekunden entgegen nimmt und als Returnwert auch genau diesen int-Wert zurückliefert. Ist der Wert für die Dauer kleiner als 0, so wird in der Servicefunktion des Servers kein sleep()-Aufruf und kein Schlafen durchgeführt sondern sofort der Returnwert zurückgegeben.
4. Funktion `vs_shutdown()`,
   über die der RPC-Server beendet werden kann. Beim Beenden soll der Server auch seine Registrierung beim Binder löschen und belegte Ressourcen freigeben.

Testen sie das Verhalten des RPC-Servers, wenn mehrere RPC-Clients gleichzeitig, d.h. konkurrent auf den RPC-Server zugreifen. Führen Sie Situationen herbei, bei denen der RPC-Server eine RPC-Funktion noch nicht beendet hat, während schon der nächste Request für genau diese RPC-Funktion beim RPC-Server eintrifft.

## *Optionale* Aufgabe 5: Multithreaded RPC-Server

Überlegen Sie, wie Sie aus Ihrem RPC-Server-Simple aus Aufgabe 4 einen Multithreading-fähigen RPC-Server (**RPC-Server-Simple-MT**) machen können.

***Hinweise*:**

1. Erzeugen Sie mit der Option `-M` bei `rpcgen` Thread-sicheren Code für die RPC-Funktionen.
   Beachten Sie: die Option -M führt nur zur Erzeugung von neuen Schnittstellen für die Servicefunktionen und Stubs, so dass diese Thread-sicher implementiert werden können. Bei der Verwendung der Option -M wird noch kein Multithreading-fähiger RPC-Server generiert.
2. Analysieren Sie den durch `rpcgen -M -a` erzeugten Code. Betrachten Sie besonders die Schnittstellen, über welche die Dispatcher-Funktion nun die Service-Funktionen aufruft. Leiten Sie daraus ab, was Sie bei der Implementierung Ihrer Service-Funktionen anders machen können und wegen des Multithreading auch anders machen müssen.
3. Die Schnittstellen der Client-Stub-Funktionen haben sich durch die Anwendung der Option `-M` geändert haben. Die Returnwerte der Interface-Funktionen aus der Schnittstellenbeschreibungsdatei (x-Datei) werden jetzt als Aufrufparameter der Client-Stub-Funktionen implementiert. Beachten Sie, dass Sie die Aufrufparameter, die nur als "Returnwerte" dienen, vor dem Aufruf der Client-Stub-Funktion auch **initialisieren** müssen.
4. Überlegen Sie, wie die Dispatcher-Funktion mit Hilfe von PThreads Multithreading-fähig gemacht werden kann.
5. Rufen Sie in der Dispatcher-Funktion an der richtigen Stelle eine Thread-Create-Wrapper-Funktion auf, die Sie selbst schreiben.
6. Duplizieren Sie in der Thread-Create-Wrapper-Funktion soweit notwendig die Aufrufdaten und packen Sie diese in eine Struktur, die Sie beim Erzeugen des Threads als Argumente für die Thread-Start-Funktion mit geben.
7. Schreiben Sie eine Thread-Start-Funktion, welche die eigentliche Service-Funktion aufruft, die Antwort an den Client zurücksendet und dann die allokierten Ressourcen wieder freigibt.
8. Geben Sie an geeigneten Stellen die PThread-ID mit `pthread_self()` aus, damit Sie erkennen können, ob die Servicefunktionen wirklich in verschiedenen PThreads laufen. Beachten Sie dabei, das die PThread-Bibliothek die PTread-IDs in der Regel wiederverwendet, wenn ein PThread vollständig beendet und gelöscht ist.
9. Implementieren Sie die Lösungen von Aufgabe 4 und Aufgabe 5 getrennt, damit Sie beide Lösungen gleichzeitig laufen lassen können und das Verhalten in beiden Lösungen vergleichen können.
10. Diese interessante, optionale Aufgabe ist für Teilnehmer mit viel freier Zeit gedacht.


 