SunRPC

KVBK - Konzepte von Betriebssystem-Komponenten

1 Einleitung

1.1 Problemstellung

In einigen Situationen ist es wünschenswert, bestimmte Programmteile (Prozeduren) auf einem entfernten System auszuführen. Dies ist zum Beispiel der Fall, wenn man ein Programm so aufteilen will, dass eine rechenintensive Berechnung auf mehreren Rechnern parallel ausgeführt wird. Durch die Einführung einer abstrakten, generischen Schnittstelle zum entfernten Prozeduraufruf (RPC, *Remote Procedure Call*) kann weiterhin auch das Problem von Multiprozessorsystemen elegant gelöst werden: die Verteilung der Aufgaben auf die verschiedenen Prozessoren hat dieselbe Syntax wie die Verteilung auf mehrere Systeme.
Durch eine RPC-Schnittstelle ist also eine Verteilung der Aufgaben, die ein Programm vollführen soll, leicht möglich.

1.2 Einsatzzweck von SunRPC

Eine solche generische Schnittstelle ist *SunRPC*. Die *SunRPC*-Spezifikation in der Version 2 wurde von der Firma Sun im August 1995 durch die RFC 1831[1][Neta] festgelegt. *SunRPC* wird eingesetzt, um eine einheitliche Schnittstelle zu schaffen, mit deren Hilfe man Prozeduraufrufe auf entfernten Systeme vergleichsweise einfach realisieren kann. Hierbei kann unter anderem auch die Architektur des Clients und Servers - und dementsprechend das Datenformat[2] verschieden sein.

2 Technische Realisierung

2.1 Aufbau von SunRPC

Jeder Server, der mithilfe von *SunRPC* Prozeduren exportiert, hat ein einheitliches Grundgerüst: der Server besteht zunächst aus verschiedenen Programmen, denen jeweils eine Nummer zugewiesen ist. Einige Programmnummern sind von Sun reserviert, andere hingegen kann der Benutzer für eigene Programme verwenden. Bekannte Programmnummern sind beispielsweise 100003, welches für NFS[3] benutzt wird oder 100004, welches dem ypserv[4] zugewiesen ist. Der Programmnummernbereich teilt sich wie folgt auf:
Jedes Programm besteht nun seinerseits aus einer oder mehreren Versionen, die wieder durchnummeriert werden. NFS ist zum Beispiel in der Version 3 momentan noch am weitesten verbreitet. Wenn man jedoch ein neueres LINUX-System einsetzt, wird man feststellen, dass dort auch schon die neuere NFS-Version 4[5][Netc] unterstützt wird. Der *SunRPC*-Dienst bietet nun auf diesem neueren System

[1] „Request for Comments“ spezifizieren viele bekannte Protokolle wie etwa HTTP, FTP, POP3, SMTP oder NNTP
[2] Systeme können sich beispielsweise in ihrer „Endianness“ unterscheiden
[3] Network File System, in der Unix-Welt wohl die gebräuchlichste Variante, Dateisysteme zwischen vernetzten Rechnern zu teilen
[4] YP steht hier für „Yellow Pages“. Der ypserv ist ein Teil von NIS, dem Network Information Service, welcher beispielsweise verwendet werden kann, um eine Benutzerdatenbank zentral zu verwalten
[5] http://www.nfsv4.org/

Programmnummer	Bedeutung
0 - 1fffffff	Von Sun definiert
20000000 - 3fffffff	Benutzerdefiniert
40000000 - 5fffffff	Temporär vergeben
60000000 - ffffffff	Reserviert

Tabelle 1: Verfügbare und reservierte Programmnummern

für das Programm 100003 (NFS) die Versionen 2, 3 und 4 an.
Jede Version wird nun ihrerseits wieder weiter unterteilt in verschiedene Prozeduren, welche auch durchnummeriert werden. Jede Version bietet grundsätzlich die Prozedur Nummer 0 an, die sogenannte „Nullprozedur“. Die Nullprozedur nimmt keine Argumente entgegen und gibt kein Resultat zurück. Sie hat dennoch einen Zweck: ein Client kann testen, ob ein RPC-Server überhaupt eine besimmte Version eines Programms unterstützt. Sollte ein NFS-Client beispielsweise die neuere NFS-Version 4 unterstützen und will herausfinden, ob der Server, mit dem er sich verbinden soll, diese auch unterstützt, geschieht folgendes: der Client versucht zuallererst die Nullprozedur auf dem Server auszuführen:

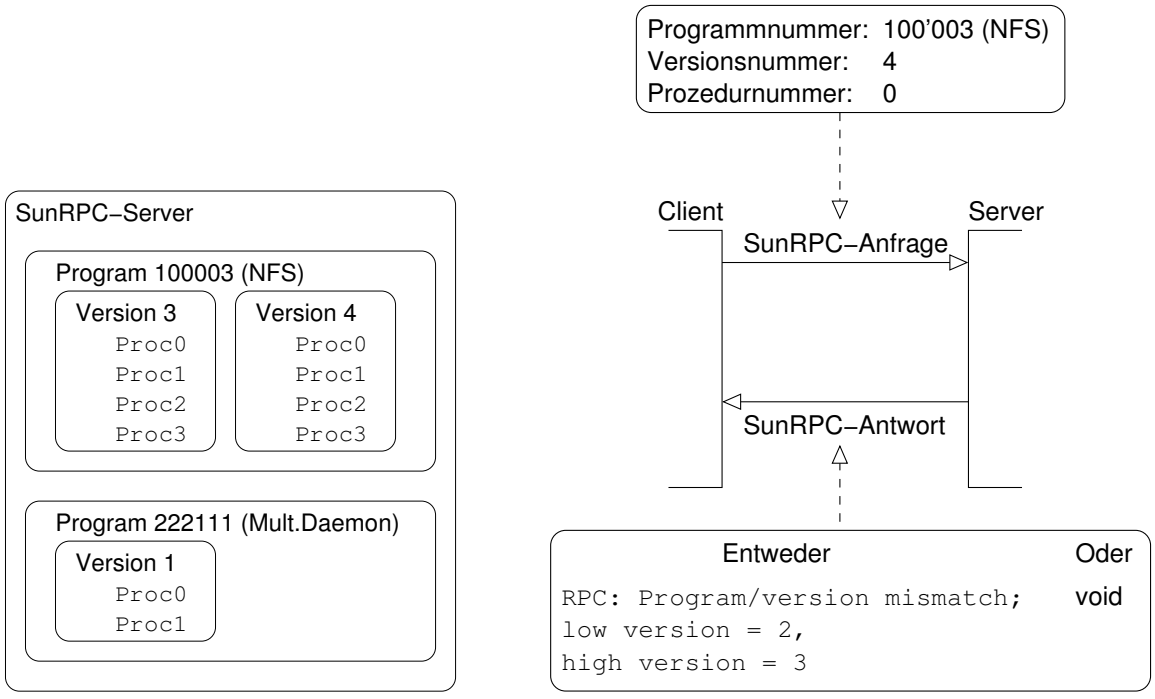

Abbildung 1: Serverarchitektur und Aufruf der Nullfunktion als PING-Test

Unterstützt nun der Server auch die Protokollversion 4, so wird er einfach nichts zurückgeben: die Nullprozedur wurde erfolgreich ausgeführt, hat aber keinen Rückgabewert (void). Wird die Protokollversion vom Server aber nicht unterstützt, so bekommt der Client beim Versuch, den call auszuführen einen Fehler: „Program/version mismatch“. Der Client wird dann automatisch auf eine niedrigere Protokollversion zurückgreifen.

2.2 Syntax und Semantik von SunRPC und XDR

Damit *SunRPC* Daten auch architekturunabhängig austauschen kann, wurde von Sun eine besondere, einheitliche Beschreibungssprache in der RFC 1832[Netb] offengelegt: *XDR*[6][ETH]. *XDR* ist eine hardwareunabhängige formale Sprache, die in der Syntax C sehr ähnelt. Wenn *SunRPC*-Aufrufe von einem System zu einem anderen erfolgen, werden die Aufrufparameter zunächst in *XDR* „verpackt“ und am Zielsystem wieder „ausgepackt“:

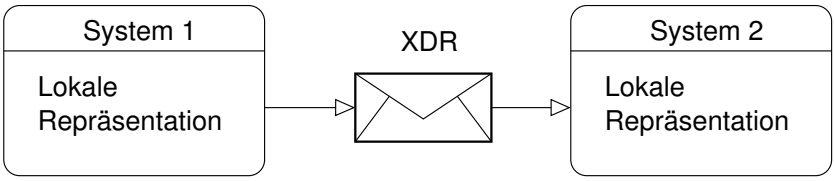

Abbildung 2: Verpackung der Daten in XDR

Durch XDR werden verschiedene architekturabhängige Kodierungen abstahiert und gekapselt:

- Endianness (Position des MSB, „Most Significant Byte“ innerhalb einer Integer-Zahl)
- Format von Gleitkommazahlen (Feste Definition der Länge der Mantisse und des Exponenten)
- Länge und Ende von Strings
- Alignment auf Wortgrenzen bei Strukturen (structs)
- Zeichendarstellung (einheitlicher Zeichensatz)

2.3 Bedeutung des Portmappers auf Netzwerkebene

Jedem Programm ist nicht nur eine Programmnummer, sondern auch eine Portnummer des Rechners zugeordnet (TCP/IP beziehungsweise UDP/IP). Da allerdings viel weniger TCP/UDP-Ports verfügbar sind als Programmnummern musste ein Konzept gefunden werden, wie ein Client, der sich zum Server mit einem bestimmten Programm verbinden will herausfinden kann, an welchem Port er das tun muss.[IBM] Hierfür wurde der Portmapper erfunden. Der Portmapper ist der einzige *SunRPC*-Dienst, der immer auf demselben „well-known“ Port läuft: 111. Wenn nun ein Client den NFS-Dienst des Servers nutzen will wird folgender Ablauf ausgeführt:

[6] eXternal Data Representation

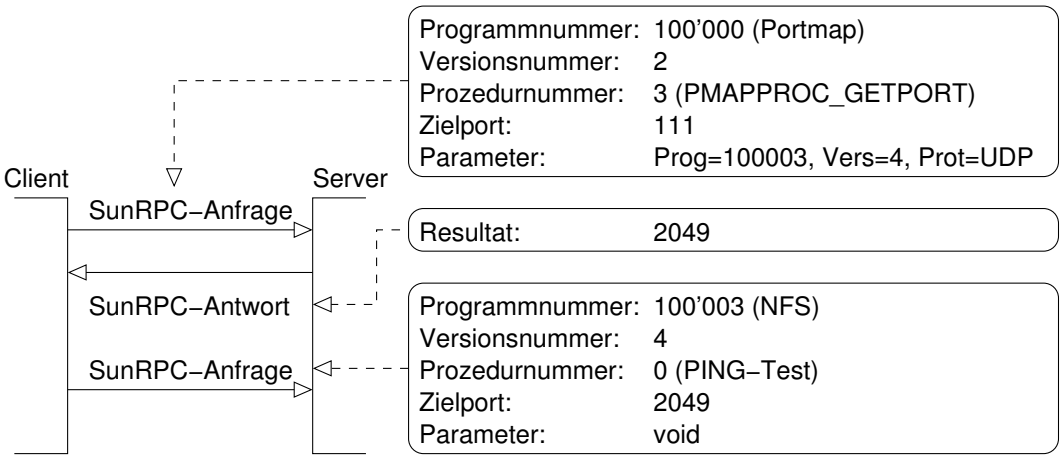

Abbildung 3: GETPORT-Anfrage an den Portmapper

2.4 Programmierung eines SunRPC Clients/Servers

Um mit einem einfachen Client/Server-Modell in SunRPC anzufangen benutzt man am Besten das Tool „rpcgen“[Hew][Zü], welches aus einer spezeillen *XDR*-Beschreibung des Clients/Servers automatisch ein Grundgerüst (Stub/Skeleton) und Beispielprogramme generiert. Wichtig ist im Hinterkopf zu behalten, dass XDR nicht alle Sprachkonstrukte, die zum Beispiel C anbietet, abbilden kann. Speicherzeiger (Pointer) sind bei einem RPC-Aufruf beispielsweise immer problematisch, da die Speicherbereiche des Quell- und Zielsystems unterschiedliche sind. Allerdings kann man Strings (Character-Pointer) in XDR ausdrücken. Auch Arrays variabler Länge werden durch das Marshaling des XDR-Präprozessors in C-konforme Konstrukte gewandelt: hier wird eine Struktur (struct) erzeugt, die in einem Integer die Länge speichert und der zugehörige Speicherbereich wird kopiert.

Ein einfaches Beispiel (Beispiel.x), wie eine XDR-Beschreibung aussehen kann:

```
1  struct l_Resultat {
2      int Ergebnis;
3  };
4
5  struct l_Parameter {
6      int Faktor1;
7      int Faktor2;
8  };
9
10 program EXAMPLE_PROG {
11     version PROGRAM_VERS {
12         l_Resultat MULTIPY(l_Parameter) = 1;     /* Prozedurnummer 1 */
13     } = 1;                                       /* Versionsnummer 1 */
14 } = 222111;                                     /* Programmnummer 222111 */
```

Dieses wird dann mit Hilfe des Kommandos „rpcgen -a Beispiel.x“ in die verschiedenen Client- und Serverdateien umgewandelt. Ohne jetzt weiter auf die Feinheiten eingehen zu wollen, soll kurz gezeigt werden, wie der Server-Quellcode aussieht (Beispiel_server.c):

```
1  #include "Beispiel.h"
2
3  l_Resultat* multipy_1_svc(l_Parameter *argp, struct svc_req *rqstp) {
4      static l_Resultat  result;
```

```
5
6       result.Ergebnis = (argp->Faktor1) * (argp->Faktor2);
7
8       return &result;
9   }
```

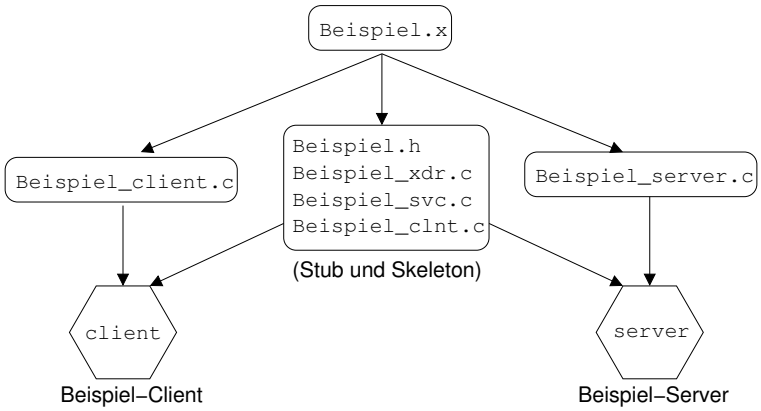

Abbildung 4: Funktionsweise von rpcgen

Man kann also deutlich erkennen, dass das Programm eine „entfernte Multiplikation“ durchführt: der Resultatwert „Resultat“ wird gleich dem Produkt der übergebenen Parameter „Faktor1“ und „Faktor2“ gesetzt. Da der Client-Code etwas umfangreicher ist (da zum Beispiel Fehlerabfragen für den Fall der Nichtausführbarkeit enthalten sein müssen) wird dieser hier nicht gezeigt, sondern nur die Funktionalität des Gesamtgebildes. Sobald der Server mit „./server &“ gestartet wurde, wird er auch schon in die Liste der RPC-Dienste eingetragen. Ein Aufruf von „rpcinfo -p“ zeigt (gekürzte Ausgabe):

```
   Program Vers Proto   Port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
    100003    2   tcp   2049  nfs
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    222111    1   udp    926
    222111    1   tcp    928
```

Hier ist ablesbar, dass unser Multiplikationsdienst mit der Programmnummer 222111 in der Version 1 auf UDP-Port 926 und auf TCP-Port 928 erreichbar ist.

2.5 Analyse von SunRPC auf Netzwerkebene

Um nun zu zeigen, dass diese Fernaufrufe in der Praxis wirklich genau so von statten gehen, kann man durch einen Netzwerksniffer (z.B. „ethereal“) auf Netzwerkebene „mithören“ und sich anzeigen lassen, was für Anfragen beziehungsweise Antworten gesendet wurden. Außerdem ist das ein guter Test für den vorher kurz gezeigten „Multiplikationsdaemon“. Hier ein Auszug des Mitschnitts (alles ab dem IP-Layer wird gezeigt):
Im ersten Paket sieht man zunächst die Anfrage an den Portmapper. Der Client (172.17.5.36) will vom Server (172.17.5.129) herausfinden, an welchem Port der Multiplikationsdaemon (222111) läuft.:

```
No.     Time        Source                Destination           Protocol Info
      1 0.000000    172.17.5.36           172.17.5.129          Portmap  V2 GETPORT Call (Reply In 2)

Internet Protocol, Src Addr: 172.17.5.36 (172.17.5.36), Dst Addr: 172.17.5.129 (172.17.5.129)
User Datagram Protocol, Src Port: 32794 (32794), Dst Port: sunrpc (111) // Zielport 111 (Portmap)
Remote Procedure Call, Type:Call XID:0x3a990561
    XID: 0x3a990561 (983106913)
    Message Type: Call (0)
    RPC Version: 2
    Program: Portmap (100000) // Angeforderte Programmnummer (100000 = portmap)
    Program Version: 2 // Angeforderte Versionsnummer (2)
    Procedure: GETPORT (3) // Angeforderte Prozedurnummer (3 = GETPORT)
    The reply to this request is in frame 2
    Credentials
    Verifier
Portmap
    Program Version: 2
    V2 Procedure: GETPORT (3)
    Program: Unknown (222111)
    Version: 1
    Proto: UDP (17)
    Port: 0
```

Der Server antwortet nun, dass dies Port 926 sei:

```
No.     Time        Source                Destination           Protocol Info
      2 0.000281    172.17.5.129          172.17.5.36           Portmap  V2 GETPORT Reply (Call In 1)

Internet Protocol, Src Addr: 172.17.5.129 (172.17.5.129), Dst Addr: 172.17.5.36 (172.17.5.36)
User Datagram Protocol, Src Port: sunrpc (111), Dst Port: 32794 (32794) // Antwort zurück an Sourceport
Remote Procedure Call, Type:Reply XID:0x3a990561
    XID: 0x3a990561 (983106913)
    Message Type: Reply (1)
    Program: Portmap (100000)
    Program Version: 2
    Procedure: GETPORT (3)
    Reply State: accepted (0) // Anfrage war in Ordnung!
    This is a reply to a request in frame 1
    Time from request: 0.000281000 seconds
    Verifier
    Accept State: RPC executed successfully (0)
Portmap
    Program Version: 2
    V2 Procedure: GETPORT (3)
    Port: 926 // Resultat: Port 926
```

Nun sendet der Client seine Anfrage, die für den Multiplikationsdaemon bestimmt ist, an den Port 926 des Servers. In der „RPC Continuation Data“ sind (obwohl hier nicht extra aufgeführt) die Zahlen 123 und 234 enthalten:

```
No.     Time        Source                Destination           Protocol Info
      3 0.000340    172.17.5.36           172.17.5.129          RPC      Continuation

Internet Protocol, Src Addr: 172.17.5.36 (172.17.5.36), Dst Addr: 172.17.5.129 (172.17.5.129)
User Datagram Protocol, Src Port: 32794 (32794), Dst Port: 926 (926) // Jetzt Verbindung auf Port 926
Remote Procedure Call
    Continuation data // Hierin sind die Daten verpackt
```

Der Server antwortet seinerseits nun in der „RPC Continuation Data“ mit dem richtigen Ergebnis, 28782:

```
No.     Time        Source                Destination           Protocol Info
      4 0.000466    172.17.5.129          172.17.5.36           RPC      Continuation

Internet Protocol, Src Addr: 172.17.5.129 (172.17.5.129), Dst Addr: 172.17.5.36 (172.17.5.36)
User Datagram Protocol, Src Port: 926 (926), Dst Port: 32794 (32794) // Antwort an Sourceport
Remote Procedure Call
    Continuation data // Hierin ist die Antowort verpackt
```

3 Abschließende Bewertung

Trotz des interessanten Ansatzes von *SunRPC* treten in der Praxis leider doch einige Probleme auf. So ist zum Beispiel *SunRPC* schon mehrfach unangenehm in CERT-Advisories[7] aufgefallen. Auch ermöglicht der Portmapper nur eine ungenügende Authentifizierung von Clients, eine Authentifizierung muss also wieder manuell von Programmierer realisiert werden, wenn *SunRPC*-Dienste auf nicht vertrauten Umgebungen (wie z.B. dem Internet) freigegeben werden sollen. Zuletzt bereitet vorallem der Portmapper-Dienst erhebliche Schwierigkeiten, wenn versucht werden soll, einen *SunRPC*-Dienst z.B. zu tunneln, da die Ports dynamisch vergeben werden. Deswegen stellt NFS in der Version 4 unter anderem ein „NFS over TCP“ bereit, das komplett über TCP/IP läuft. Dieses ist erheblich „pflegeleichter“ und hat praktisch keine Performancedefizite gegenüber NFS3, welches *SunRPC* benutzt.

Literatur

[ETH] ETH Zürich, Prof. Dr. Mattern, Friedmann. Verteilte Systeme, WS 2003/2004. *[Internet]*. URL: http://www.vs.inf.ethz.ch/edu/WS0304/VS/slides/Vorl.VertSys03_04_7.pdf.

[Hew] Hewlett Packard. Writing RPC Applications with the rpcgen Protocol Compiler. *[Internet]*. URL: http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/HTML/AA-Q0R5B-TET1_html/onc-rpc3.html.

[IBM] IBM. Communication Programming Concepts: RPC Port Mapper Program. *[Internet]*. URL: http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/aixprggd/progcomc/rpc_ref.htm.

[Neta] Network Working Group, R. Srinivasan, Sun Microsystems. RFC 1831: RPC Remote Procedure Call Protocol Specification Version 2. *[Internet]*. URL: http://www.ietf.org/rfc/rfc1831.txt.

[Netb] Network Working Group, R. Srinivasan, Sun Microsystems. RFC 1832: XDR: External Data Representation Standard. *[Internet]*. URL: http://www.ietf.org/rfc/rfc1832.txt.

[Netc] Network Working Group, S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, Sun Microsystems, Inc., C. Beame, Hummingbird Ltd., M. Eisler, D. Noveck, Network Appliance, Inc. RFC 3530: Network File System (NFS) version 4 Protocol. *[Internet]*. URL: http://www.ietf.org/rfc/rfc3505.txt.

[Zü] Züricher Hochschule Winterthur, Feisthammel Patrick. Softwaresysteme/Verteilte Systeme, WS 2003/2004. *[Internet]*. URL: http://www-t.zhwin.ch/ fei/sosy/rpc/2-rpc_a4.4fach.pdf.

[7] Analyse und Dokumentation von Sicherheitslücken von Softwaresystemen. Siehe http://www.cert.org/advisories/