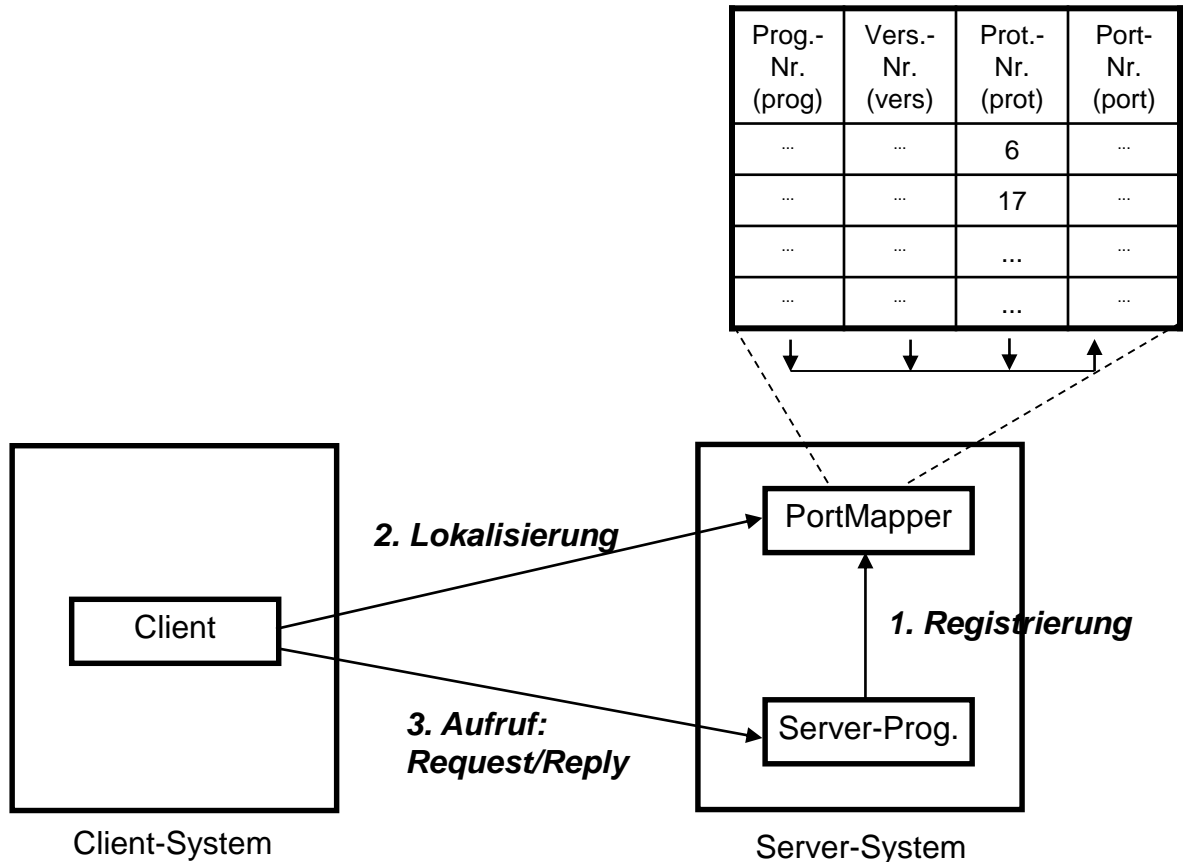


"single thread of control accross a network"

RPC: Programm-Modell

Name	RPC-Programm-Nummer(n)	Beschreibung
portmapper	00100000	Portmapper, RPCBIND
nfs	00100003	Network File System
yellow pages	00100004	NIS (Network Information Service)
mount demon	00100005	Mount Protocol im NFS3
ether stats	00100010	Ethernet Statistics
lock manager	00100021	Lock Manager im NFS3
Bereiche:	00000000	reserviert
	00000001 - 1FFFFFFF	verwaltet von der IANA (Internet Assigned Numbers Authority)
	20000000 - 3FFFFFFF	benutzer-definiert, d.h. freigegeben für Verwendung durch Anwendungsprogrammierer
	40000000 - 5FFFFFFF	transiente (d.h. temporäre) Nummern: für Anwendungen, die dynamisch Programmnummern generieren
	60000000 - FFFFFFFF	nicht vergeben

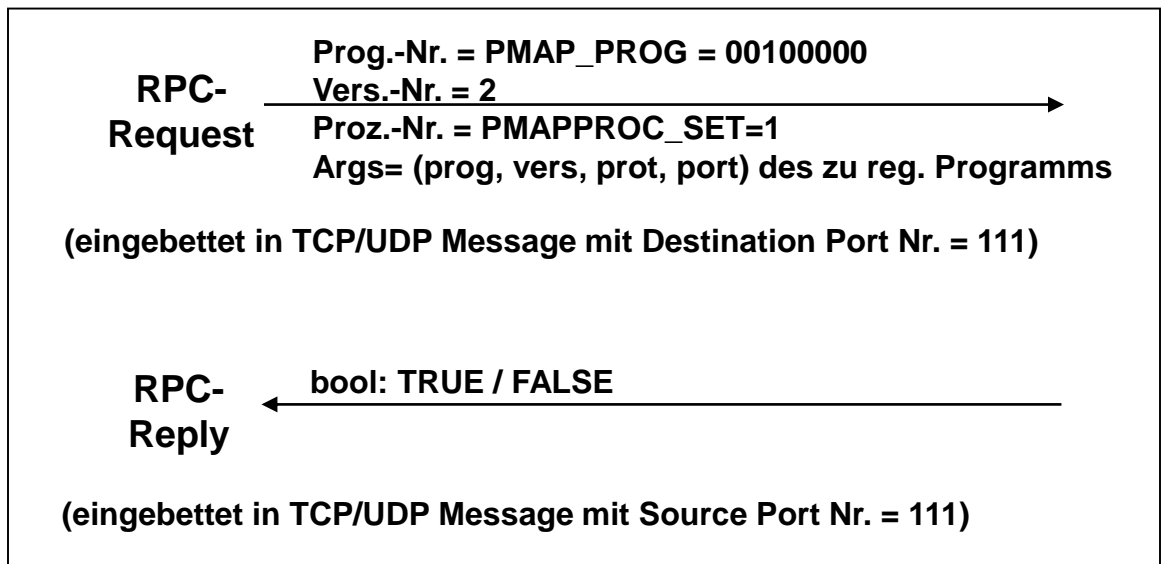
www.iana.org



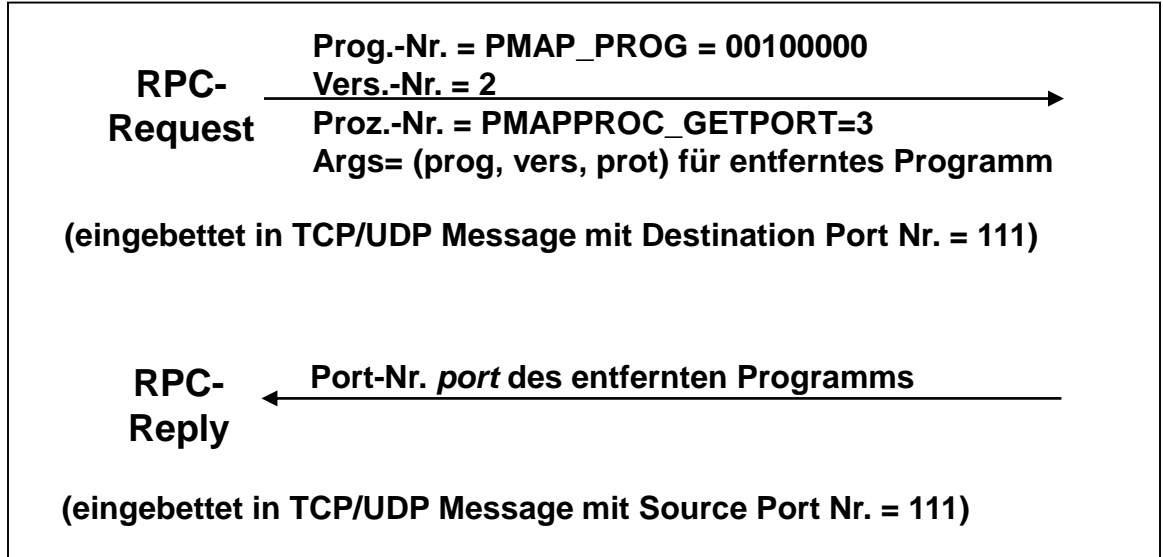
Registrierung, Lokalisierung und Benutzung eines Remote Programs auf dem Server System:

- Ein Server-Programm mit Prozeduren registriert sich als Remote Program beim lokalen Port Mapper (RPC prog number, version number, protocol number (TCP=6, UDP=17), port number).
- Der Client sendet eine Lokalisierungsanfrage mit Programm-, Versions- und Protokoll-Nummer zum Portmapper auf dem Server-System.
- Der Portmapper schickt die Port-Nummer des spezifizierten Remote Programs zurück an den Client.
- Der Client sendet ein Request mit Programm-, Versions- und Prozedur-nummern an den Port mit der erhaltenen Portnummer der Server-Maschine.
- Das Remote Program schickt ein Resultat zurück an den Client.

Registrierung am Portmapper



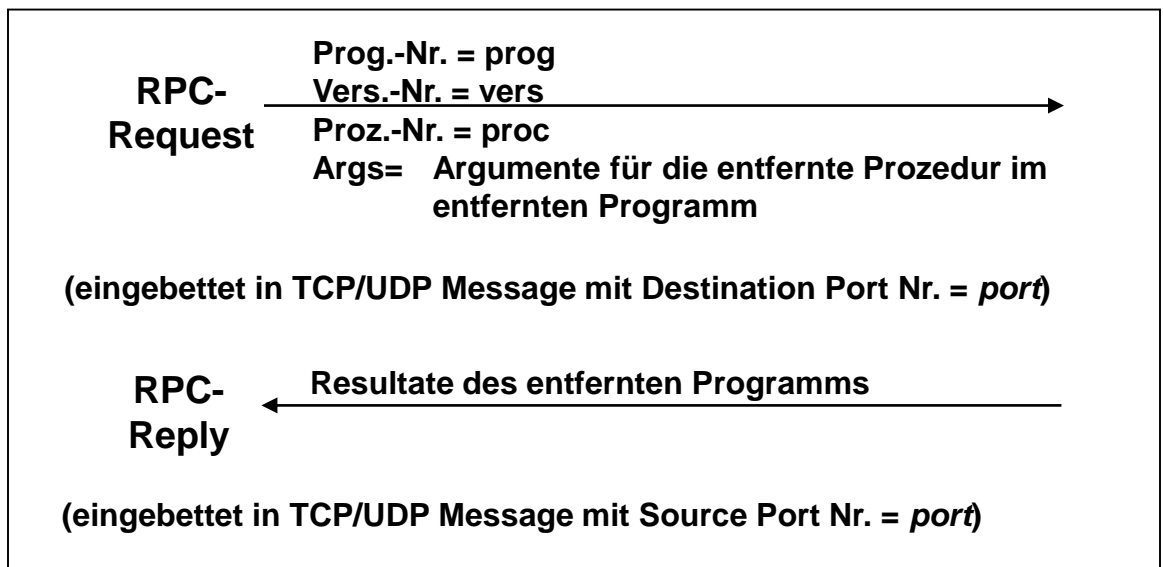
Lokalisierung am Portmapper

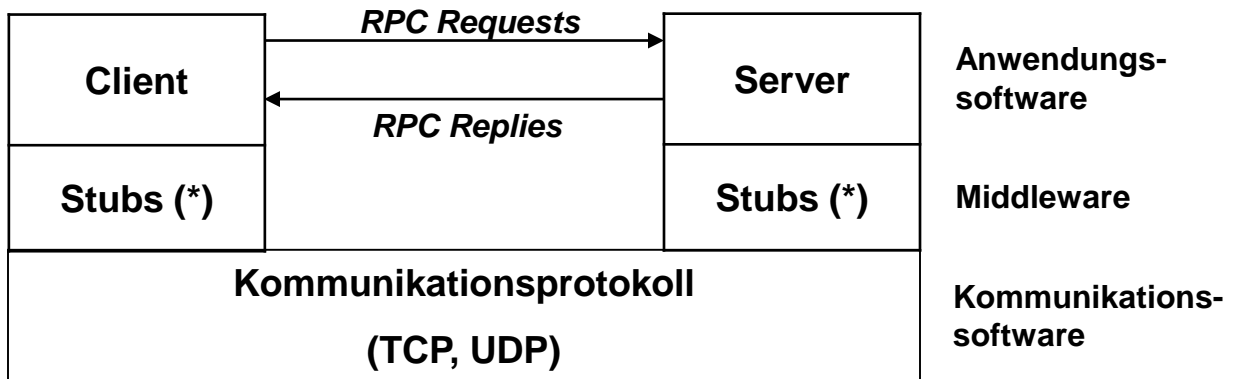


... jetzt kann die entfernte Prozedur im entfernten Programm unter der Portnummer *port* vom Client aus aufgerufen werden.

RPC: Ablaufmodell II

Aufruf der entfernten Prozedur *proc* am Port Nr. *port*:





(*) Interface Stub und Communication Stub

Integer:

XDR Data Type Declaration:

int identifier;

XDR Data Type Encoding:

byte 0	byte 1	byte 2	byte 3
--------	--------	--------	--------

(Enums werden wie Integers codiert: 4 Bytes)

String:

XDR Data Type Declaration:

string identifier <n>;

*variable Länge,
maximal n Bytes*

*Padding auf
Vielfaches
von 4 Bytes*

XDR Data Type Encoding:

length n	byte 0	byte 1	byte n-1	0	0
----------	--------	--------	-------	----------	---	-------	---

(4 Bytes)

(Opaque Daten werden analog codiert: Länge, Bytes und Padding)

[Quelle: RFC 4506]

XDR: Datentypen und Codierungen I

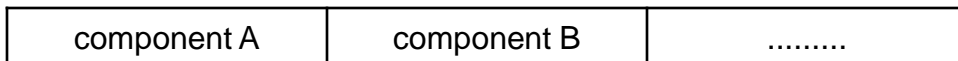
VS 2.7

Structure:

XDR Data Type Declaration:

struct {component A, component B, ... } identifier;

XDR Data Type Encoding:

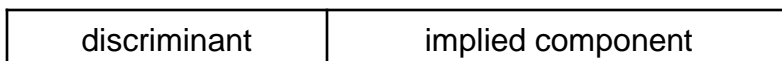


Union:

XDR Data Type Declaration:

**union switch (discriminant) {
 case discriminant value-A: declaration-A;
 case discriminant-value-B: declaration-B;
 ...
} identifier;**

XDR Data Type Encoding:



[Quelle: RFC 4506]

XDR-Datentyp	Beschreibung
int	32-bit signed binary integer
unsigned int	32-bit unsigned binary integer
enum	Enumeration type
bool	Boolean (false (0) bzw. true (1) (spezieller enum))
float	single precision floating point (32 bit)
double	double precision floating point (64 bit)
opaque	uninterpretierte Daten: feste: [...] oder variable Länge: <..>
string	Folge von ASCII-Bytes, variable Länge: <..>
array	Array mit beliebigem Datentyp: feste: [...] oder variable Länge: <..>
struct	Struktur mit Elementen u.U. verschiedener Typen
union	Datenstruktur mit alternativen Komponenten
<i>etc.: hyper (unsigned) int, quadruple, void, constant, typedef, optional-data.</i>	

Routine (Argumente)	codierter Datentyp
xdr_int (xdrs, ip)	int
xdr_u_int (xdrs, up)	unsigned int
xdr_enum (xdrs, ep)	enum
xdr_bool (xdrs, bp)	bool
xdr_float (xdrs, fp)	float
xdr_double (xdrs, dp)	double
xdr_opaque (xdrs, op, count)	opaque (variable Länge)
xdr_string (xdrs, sp, maxsize)	string (variable Länge)
xdr_vector (xdrs, ap, size, elsize, elproc)	array (feste Länge)
xdr_union (xdrs, discrip, unproc, choices, default)	union
<i>etc., z.B. xdr_array() für Arrays variabler Länge</i>	

1. Argument: Zeiger auf XDR Stream
2. Argument: Zeiger auf Speicher mit/für XDR-Daten
3. Argument: siehe Manuals

XDR: Datentypen und Codierungs-Routinen VS 2.9

```

const MAXUSERNAME = 32;
const MAXFILELEN = 65536;
const MAXNAMELEN = 255;

enum filekind {TEXT=0, DATA=1, EXEC=2};

union filetype switch (filekind kind) {
    case TEXT: void;
    case DATA: string creator<MAXNAMELEN>;
    case EXEC: string interpretor<MAXNAMELEN>;
};

struct file {
    string filename<MAXNAMELEN>;
    filetype type;
    string owner<MAXUSERNAME>;
    opaque data<MAXFILELEN>;
};

```

XDR Language

[RFC 4506]

Ausführbares LISP-Programm *sillyprog* des Owner *John* mit Inhalt (*quit*)

Offset	Hex Bytes	ASCII	Comments
0	00 00 00 09	Dateinamenlänge = 9
4	73 69 6c 6c	sill	Dateiname
8	79 70 72 6f	ypro	Dateiname (Fortsetzung)
12	60 00 00 00	g...	Dateiname (Ende + Padding)
16	00 00 00 02	Dateiart EXEC
20	00 00 00 04	Interpretornamenlänge = 4
24	6c 69 73 70	lisp	Interpretorname
28	00 00 00 04	Ownerlänge = 4
32	6a 6f 68 6e	joh n	Ownername
36	00 00 00 06	Datenlänge = 6
40	28 71 75 69	(qui	Daten
44	74 29 00 00	t) ..	Daten (Ende + Padding)

[RFC 4506]

XDR-Codierung (Beispiel)

```
enum msg_type { CALL = 0, REPLY = 1 };
```

```
struct rpc_msg {
    unsigned int xid;
    union switch (msg_type mtype) {
        case CALL: call_body cbody;
        case REPLY: reply_body rbody;
    } body;
};
```

```
struct call_body {
    unsigned int rpcvers;
    unsigned int prog;
    unsigned int vers;
    unsigned int proc;
    /* authentication: opaque data */
    /* remote procedure-specific call parameters */
};
```

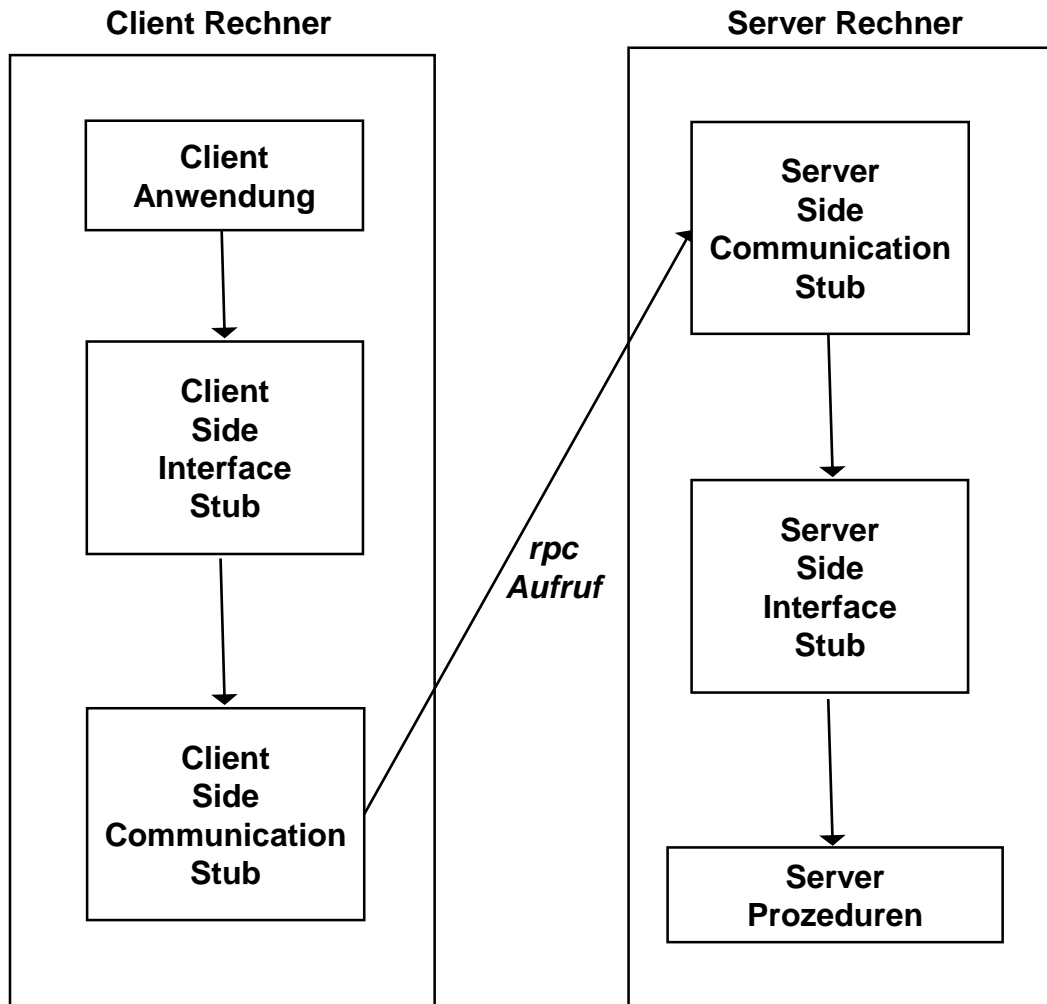
0

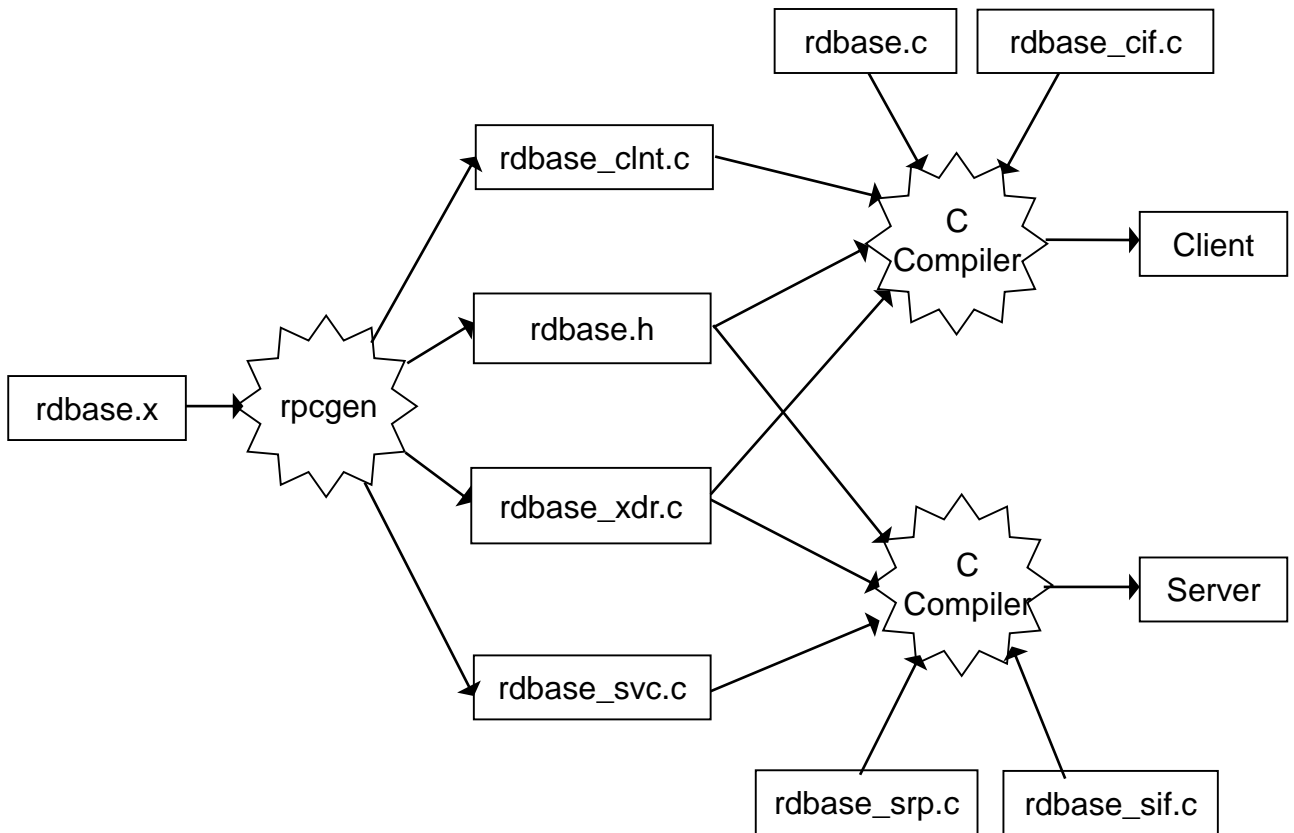
31

Transaction ID (Korrelation Request-Reply)
Message Type (0 für CALL, 1 für REPLY)
RPC Versionsnummer
Remote Program Nummer
Remote Program Versionsnummer
Remote Procedure Nummer
Authentication
Argumente , in <i>struct</i> mit u.U.mehreren Komponenten

```
struct reply_body {
    /* Inhalt: Resultate der aufgerufenen entfernten Prozedur */
};
```

RPC Messages





Eingabe für `rpcgen`:

- Spezifikationsdatei *rdbase.x* (Konstantendeklarationen, globale Datentypen, globale Daten, Remote Procedures, Argument- und Resultattypen)

Ausgaben von `rpcgen`:

- Client-side und Server-side Communication Stubs (*rdbase_clnt.c*, *rdbase_svc.c*),
- XDR-Prozeduren zur Codierung / Encodierung und zur Erzeugung von Argumenten / Resultaten (*rdbase_xdr.c*),
- Header-File für Typen und Konstanten, die auf Client- und Server-Seite verwendet werden (*rdbase.h*).

Eingaben für C-Compiler :

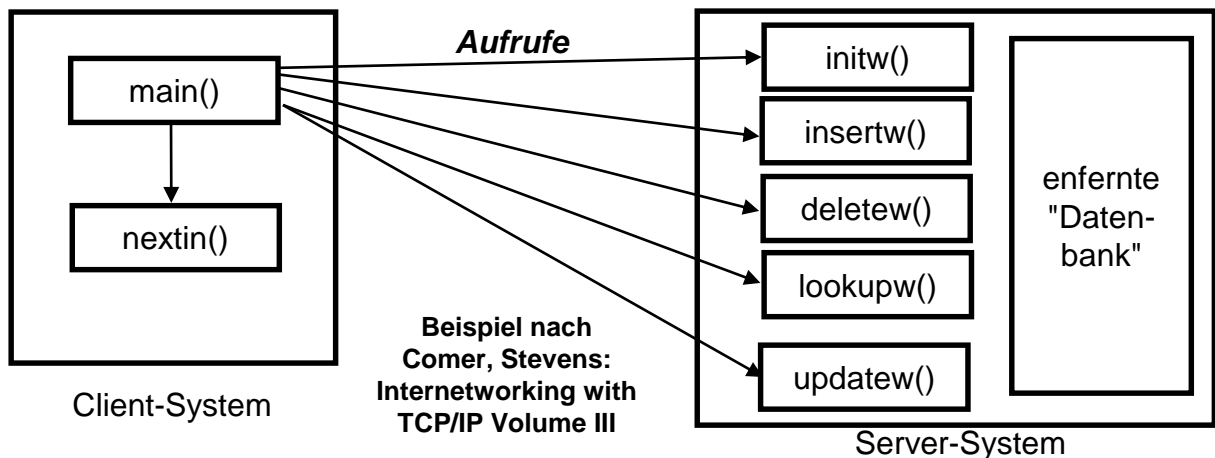
- `rpcgen`-generierte Dateien,
- Client-side und Server-side Interface Stubs (*rdbase_cif.c*, *rdbase_sif.c*),
- Client-Programm und Server-Prozeduren (*rdbase.c* und *rdbase_srp.c*).

rpcgen-Konzept

Datei	Erläuterung
<i>rdbase.h</i>	enthält Deklarationen der in der Spezifikationsdatei deklarierten Konstanten und Datentypen und die externen Deklarationen der Schnittstellenprozeduren für die Server-Prozeduren (<i>insertw_1()</i> etc.). Sie rufen dann die Remote-Prozeduren selbst auf (also z.B. <i>insertw_1()</i> ruft <i>insertw()</i> auf.).
<i>rdbase_xdr.c</i>	enthält die Aufrufe der von einer Anwendung benötigten XDR-Konversions-Routinen
<i>rdbase_clnt.c</i>	enthält den Client Side Communication Stub , bestehend aus den RPC-konformen Aufrufen <i>initw_1()</i> , <i>insertw_1()</i> , <i>deletew_1()</i> , <i>lookupw_1()</i> . Sie enthalten jeweils einen Aufruf von <i>clnt_call()</i> .
<i>rdbase_svc.c</i>	enthält den Server Side Communication Stub . Zunächst läuft ein <i>main()</i> , das: sich einen Port holt (<i>svctcp_create()</i> oder <i>svcudp_create()</i>), sich beim Portmapper registriert (<i>svc_register()</i>) und dann auf RPC-Calls wartet (<i>svc_run()</i>). Außerdem ist der Dispatcher implementiert, der die Server-Interface-Routinen aufruft (<i>switch (rqstp->rq_proc)</i>). In diesem Switch enthält der Funktionszeiger <i>local</i> einen Zeiger auf die gewünschte Anwendungsfunktion, z.B. <i>local = (...) insertw_1_svc</i> ; Anschließend wird die gewünschte Anwendungsfunktion aufgerufen: <i>result = (*local) (&argument,rqstp)</i> ;

Kommando	Argument(e)	Bedeutung
I	-	Initialisierung einer leeren "Datenbank"
i	Wort	Wort in "Datenbank" einfügen
d	Wort	Wort aus "Datenbank" löschen
l	Wort	Wort in "Datenbank" auffinden
u	Wort1 Wort2	Wort1 durch Wort2 ersetzen
q	-	Beenden des Anwendungsprogramms

- (1) Programmierung des konventionellen Programms (*main()* und Prozeduren)
- (2) Verteilung des konventionellen Programms in lokale und remote Prozeduren auf Client- bzw. Server-System



- (3) Erstellen der Rpcgen-Spezifikationsdatei, z.B. *rdbase.x*
- (4) Generierung von Stubs, XDR-Prozeduren und Header-File durch rpcgen:

rpcgen rdbase.x

- (5) Erstellen der Stub Interface Procedures auf Client- und Server-Seite
- (6) Erstellen des lauffähigen Servers: Übersetzen und Linken
- (7) Erstellen des lauffähigen Clients: Übersetzen und Linken
- (8) Starten von Server und Client auf ihren jeweiligen Maschinen (in dieser Reihenfolge):

./rdbased &
./rdbase

(für den Server),
(für den Client).

RPC-Beispiel

/* Konstanten und Datenstrukturen in Client und Server */

const MAXWORD = 50;

const DICTSIZE = 100;

struct upd {

string upd_old <MAXWORD>;

string upd_new <MAXWORD>;

};

/* Schnittstellen-Spezifikation des RPC-Programms */

/* die Sprache ist "RPC language", nicht C */

program RDBASEPROG {

version RDBASEVERS {

int INITW (void) = 1; /* Prozedur Nr. 1 */

int INSERTW (string) = 2; /* Prozedur Nr. 2 */

int DELETEW (string) = 3; /* Prozedur Nr. 3 */

int LOOKUPW (string) = 4; /* Prozedur Nr. 4 */

int UPDATEW (upd) = 5; /* Prozedur Nr. 5 */

} = 1; /* Prog.- Version Nr. 1 */

} = 0x23456789; /* Programm Nr. 0x... */

RPC-Spezifikationsdatei für verteilte Anwendung


```
#include <stdio.h>
#include <ctype.h>
#define MAXWORD 50
#define MAXLINE 80
```

```
int main (argc, argv)
int argc ;
char *argv[];
{
```

```
    char word[MAXWORD+1];
    char cmd;
    int wrdlen;
```

```
    while (1) {
```

```
        wrdlen = nextin (&cmd, word, word2);
        switch (cmd) {
            case 'l';
                initw(); break;
            case 'i';
                insertw (word); break;
            case 'd';
                if deletew (word) .... else ....; break;
            case 'l';
                if lookupw (word) .... else ....; break;
            case 'u':
                if (updatew (word, word2) .... else ....; break;
            case 'q';
                exit(0); break;
            default:
                ...
        }
    }
```

```
#include <rpc/rpc.h>
#include "rdbase.h"

#define RMACHINE "localhost"
CLIENT *handle;
```

```
handle = clnt_create
    (RMACHINE, RDBASEPROG, RDBASEVERS, "tcp");
if (handle == 0) { ...}
```

```
int nextin (cmd, word, word2)
char *cmd, *word, *word2;
{
```


```
    char command[MAXLINE];
    char tmp[10];
```

```
    gets (command, MAXLINE, stdin); /* Einlesen Kommando und Argumente */
    sscanf (command, "%s %s %s", tmp, word, word2);
```

```
    /* Übertragen in Einzelkomponenten */
    *cmd = tmp[0]; /* Übertragen Kommando */
    return 0;
```

```
}
```

RPC-Client: Beispiel



```
#include <rpc/rpc.h>
```

```
#include "rdbase.h"
```

```
#define MAXWORD 50
#define DICTSIZE 100
char dict [DICTSIZE] [MAXWORD+1];          /* die "Datenbank" */
int nwords=0;                               /* Anzahl der Worte in der "Datenbank" */
```

```
int initw() {
    nwords = 0;
    return 1;
}
```

```
int insertw (word)
char *word;
{
    strcpy (dict[nwords], word);
    nwords++;
    return nwords;
}
```

```
int deletew (word)
char *word;
{
    int i
    for (i=0; i<nwords; i++) {
        if (strcmp (word, dict[i]) == 0) {
            nwords--;
            strcpy (dict[i], dict[nwords]);
            return 1;
        }
    }
    return 0;
}
```

```
int lookupw (word)
char *word;
{
    int i
    for (i=0; i<nwords; i++) {
        if (strcmp (word, dict[i]) == 0) return 1;
    }
    return 0;
}
```

```
int updatew (word, word2)
char *word, *word2;
{
    int i;
    for (i=0; i<nwords; i++)
        if (strcmp (word, dict[i]) == 0) {
            strcpy (dict[i], word2);
            return 1;
        }
    return 0;
}
```

RPC-Server-Prozeduren: Beispiel

VS 2.18

```

/* Datei rdbase_cif.c - Client Side Interface Stub */
#include <rpc/rpc.h>
#include<stdio.h>
#include"rdbase.h"
/* die ..._1()-Prozeduren sind im rpcgen-generierten */
/* Client Side Communication Stub definiert */

```

```

extern CLIENT *handle;
static int *ret;

```

```

int initw() {
    ret = initw_1 (0, handle);
    return ret==0 ? 0 : *ret;
}

int insertw (word)
char *word ;
{
    char **arg;
    arg = &word;
    ret = insertw_1(arg, handle)
    return ret==0 ? 0 : *ret;
}

int deletew (word)
char *word ;
{
    char **arg;
    arg = &word
    ret = deletew_1(arg, handle);
    return ret==0 ? 0 : *ret;
}

```

```

int lookupw (word)
char *word;
{
    char **arg;
    arg = &word;
    ret = lookupw_1(arg, handle);
    return ret==0 ? 0 : *ret;
}

int updatew (word, word2)
char *word, *word2;
{
    struct upd arguments, *arg;
    arguments.upd_old = word;
    arguments.upd_new = word2;
    arg = &arguments;
    ret = updatew_1(arg, handle);
    return ret==0 ? 0 : *ret;
}

```

/ Erläuterungen:*

- Der Client ruft die Funktionen `initw()`,... auf, diese werden in die `rpcgen`-konformen äquivalenten Aufrufe von `initw_1()`,... umgewandelt. Damit kann der Anwender seine Aufrufe beliebig gestalten, und die eigentliche Kommunikation zwischen Client und Server läuft trotzdem `RPC`-konform.
- Die `rpcgen`konformen Aufrufe haben zwei Argumente: a) `arg`, das die Argumente der ursprünglichen Client-Funktion erhält, b) `handle`, das zur Kommunikation mit dem Server benutzt wird. **/*

Client Side Interface Stub

/ Datei rdbase_sif.c - Server Side Stub Interface */*

#include <rpc/rpc.h>

#include "rdbase.h"

/ die Prozeduren initw(), insertw(), ...sind in den */*
/ Server-Prozeduren definiert */*

static int retcode;

*int *initw_1_svc(dummy, handle)*
*void *dummy;*
*struct svc_req *handle;*
{
retcode = initw();
return &retcode;
}

*int *insertw_1_svc(w, handle)*
*char **w;*
*struct svc_req *handle;*
{
*retcode = insertw(*w);*
return &retcode;
}

*int *deletew_1_svc(w, handle)*
*char **w;*
*struct svc_req *handle;*
{
*retcode = deletew(*w);*
return &retcode;
}

*int *lookupw_1_svc(w, handle)*
*char **w;*
*struct svc_req *handle;*
{
*retcode = lookupw(*w);*
return &retcode;
}

*int *updatew_1_svc (s, handle)*
*struct upd *s;*
*struct svc_req *handle;*
{
retcode = updatew (s->upd_old, s->upd_new);
return &retcode;
}

Server Side Interface Stub

VS 2.20

rpcgen rdbase.x

***cc -c rdbase_clnt.c
cc -c rdbase_cif.c
cc -c rdbase.c***

cc -c rdbase_xdr.c

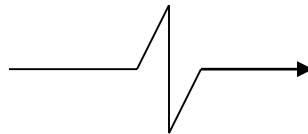
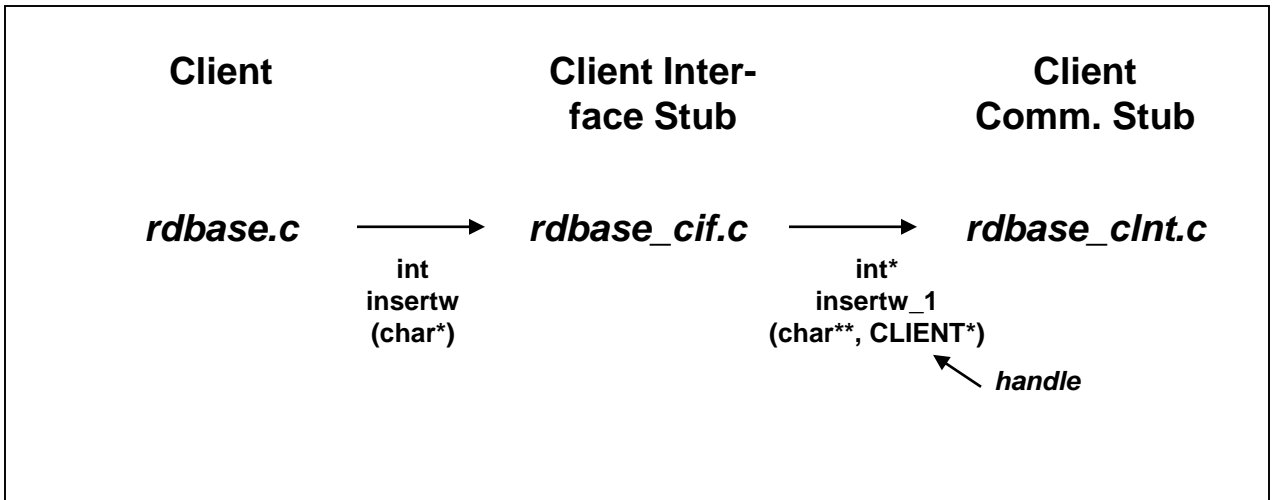
cc -o rdbase rdbase_clnt.o rdbase_cif.o rdbase.o rdbase_xdr.o

***cc -c rdbase_svc.c
cc -c rdbase_sif.c
cc -c rdbase_srp.c***

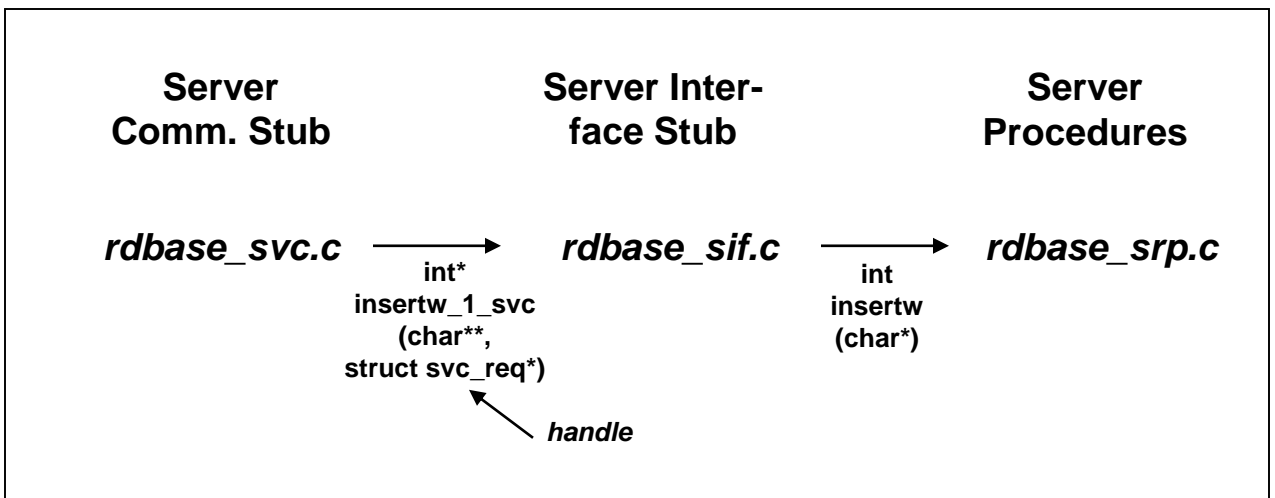
cc -o rdbased rdbase_svc.o rdbase_sif.o rdbase_srp.o rdbase_xdr.o

<i>./rdbased &</i>	<i>// auf Server-Maschine</i>
<i>./rdbase</i>	<i>// auf Client-Maschine</i>

RPC Client- und Server-Erzeugung und -Ausführung



char* clnt_call (Prozedur-Nr., Argumente, ...)
 Prog.-, Vers.- und Prozedurnummer in der RPC-Nachricht
 Portnummer im TCP-Segment bzw. UDP-Datagramm
 logische Adressen im IP-Datagramm
 physikalische Adressen im Layer-2-Frame, z.B. Ethernet-Frame



RPC: Aufruf-Abfolge