

Verteilte Systeme

5. Unicast

Unicast

The diagram illustrates unicast communication. It features two horizontal lines representing communication channels. The top line is labeled 'Sender' and has a small grey square marker. The bottom line is labeled 'Empfänger' and also has a small grey square marker. A diagonal arrow points from the 'Sender' marker down to the 'Empfänger' marker, indicating a message being sent from the sender to the receiver. Additionally, there are horizontal arrows extending to the right from both the 'Sender' and 'Empfänger' markers, representing outgoing or incoming communication paths.

Senden einer Nachricht an einen Empfänger

Unzählige Varianten

Versteckte Übertragung zusätzlicher Information

- z.B. Lampportzeit, Zeitvektor, Quittung, ...
- Unterscheidung
 - Basis- oder Anwendungsnachrichten
 - Kontrollnachrichten

Verteilte Systeme, Wintersemester 2000/2001 Folie 5.2

Synchronität

Kopplungsgrad zwischen Sender und Empfänger

Synchrone Kommunikation

- Sender blockiert bis zum Empfang der Nachricht
- Einschränkung in der Parallelität
- Weniger Pufferplatz
- Obere Schranken für Nachrichtenübertragung
- "Blocking Send"

Asynchrone Kommunikation

- Sender "blockiert" bis Nachricht kopiert wurde
- Viele Nachrichten von einem Sender kurz hintereinander (Congestion)
- Pufferung von Nachrichten
- Unbestimmte Übertragungszeit
- "Non-blocking Send"

Verteilte Systeme, Wintersemester 2000/2001 Folie 5.3

Kommunikationsmuster

Struktur der "kleinsten" Kommunikationseinheit

- Modellvorgabe

Grundmuster

- Mitteilung
 - Einzelne Nachricht vom Sender zum Empfänger
- Auftrag
 - Auftrag zum Empfänger
 - Antwort zurück an den Absender

Weitere Muster

- Multicast
- Broadcast

kriegen wir später

Verteilte Systeme, Wintersemester 2000/2001 Folie 5.4

Elementare Kommunikationsformen

	Asynchron	Synchron
Mitteilungsorientiert	Datagramm	Rendezvous
Auftragsorientiert	Asynchroner Entfernter Dienstaufruf	Synchroner Entfernter Dienstaufruf

Verteilte Systeme, Wintersemester 2000/2001 Folie 5.5

Datagramm

Sender Empfänger

send () receive ()

send () receive ()

Entkopplung von Sender und Empfänger

- Parallelarbeit möglich

Pufferung im Nachrichtensystem

- Aufwendig (Überläufe, ...)
- Blockade des Senders nur bei Speicherengpässen

Beispiele

- UDP (User Datagram Protocol)
Teil von TCP/IP (Internet)
Best Effort
max. 64 Kbyte Daten
- Signale
Einfacher Kommunikationsmechanismus in UNIX
Keine Daten oder max. 4 Byte

Sender weiß nichts

Verteilte Systeme, Wintersemester 2000/2001 Folie 5.6

Rendezvous

Sender und Empfänger sind für die Zeit der Nachrichtenübertragung gekoppelt

- Eingeschränkte Parallelität

Keine Pufferung

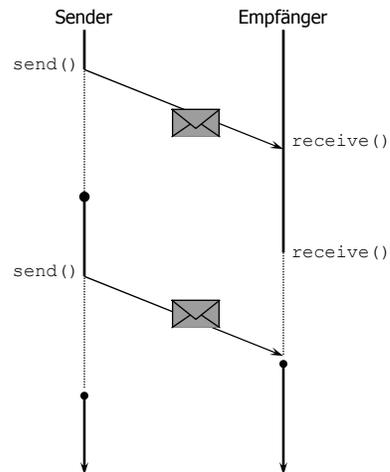
- Direkte Datenübertragung
- Keine Pufferung
- Einfachere Implementierung

Beispiele

- Ada
- QNX
- ...

Sender weiß ...

- Nachricht angekommen



Synchroner entfernter Dienstaufwurf

Synchronous Remote Service Invocation (SRSI)

Keine Parallelität

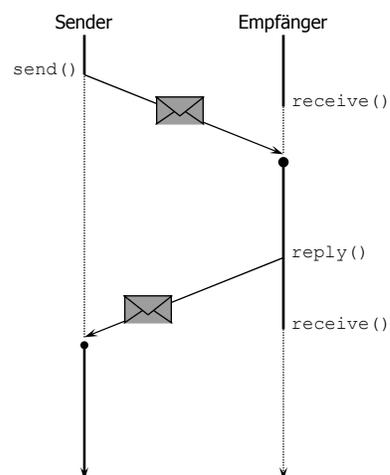
Bidirektionale Kommunikation ohne Pufferung

Beispiel

- Remote Procedure Call (RPC)
DCE, Corba, ...
- QNX

Sender weiß ...

- Auftrag angekommen
- Auftrag bearbeitet
- Resultat



Asynchroner entfernter Dienstaufwurf

Asynchronous Remote Service Invocation (ARSI)

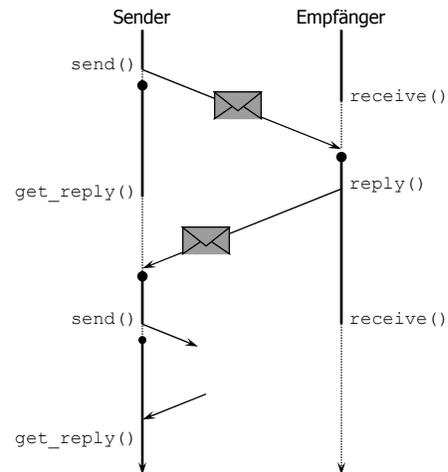
- Parallelität möglich
- Bidirektionale Kommunikation mit Pufferung
- Asynchroner RPC

Beispiel

- V-Kernel

Sender weiß ...

- Auftrag angekommen
- Auftrag bearbeitet
- Resultat



Verteilte Systeme, Wintersemester 2000/2001

Folie 5.9

Klassifikationskriterien

Elementare Kommunikationsmuster

- Synchron / Asynchron
- Mitteilungs- und Auftragsorientiert

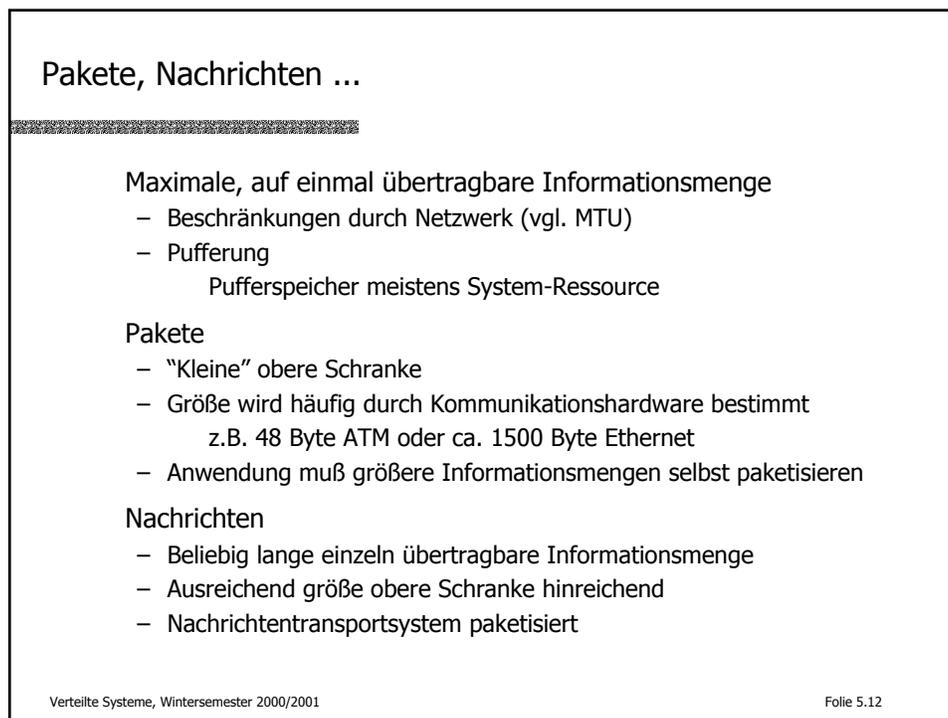
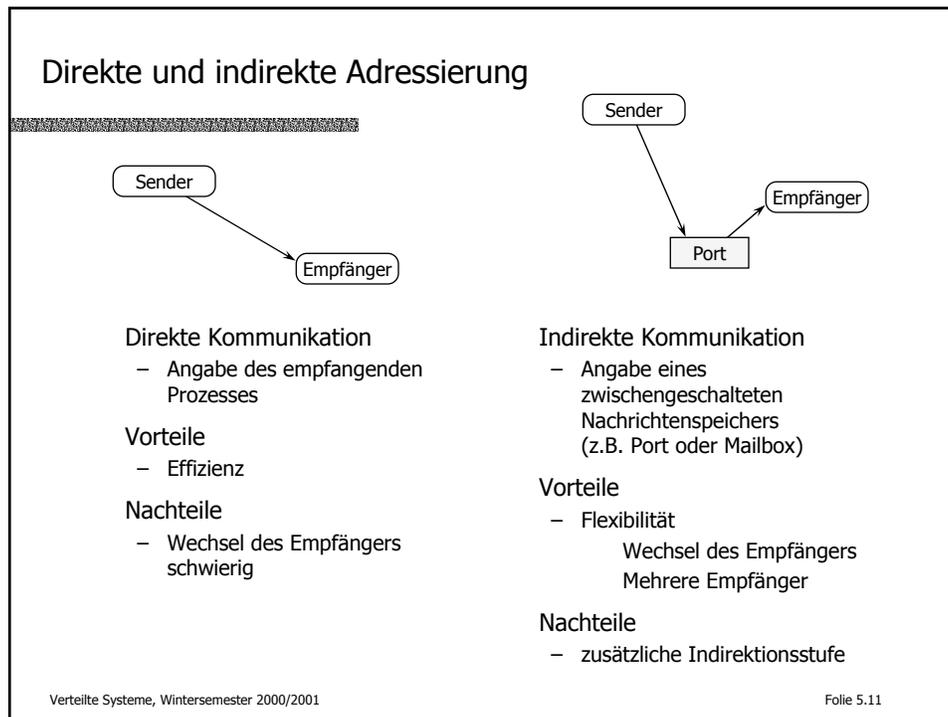
Zusätzliche Kriterien

- Direkte und indirekte Adressierung
- Pakete / Nachrichten / Ströme
- Verbindungsorientiert / Verbindungslos
- Netztopologie
 - Bus (Multicast und Broadcast)
 - Dedizierte Leitungen
- Netzausdehnung (bzgl. Bandbreite, Fehlerrate, Latenz, ...)
- ...



Verteilte Systeme, Wintersemester 2000/2001

Folie 5.10



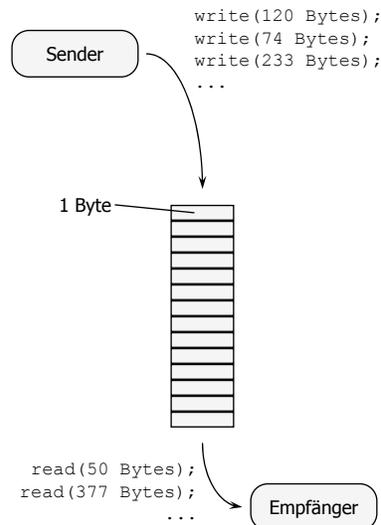
... und Ströme

Paket- und Nachrichtengrenzen
für Sender und Empfänger
unsichtbar

- "unbeschränkte" sequentiell
schreib- und lesbare Datei
- Zugriff über Read und Write
- Meist Verbindungsorientiert

Beispiel

- Pipes in UNIX
(vgl. Betriebssystemvorlesung)
- TCP



Verteilte Systeme, Wintersemester 2000/2001

Folie 5.13

5.1 Entfernter Dienstaufwurf (RPC = Remote Procedure Call)

RPC- oder IDL-Compiler

Service = Menge von entfernt aufrufbaren Prozeduren

Service-Beschreibung (Interface Description)

- Angabe der Signatur jeder Prozedur
Anzahl, Typ und Reihenfolge der Argumente
Prozedurname
Typ des Rückgabewertes
- Verwendete Sprache: "Interface Description Language" (IDL)

RPC- oder IDL-Compiler generiert

- Client-Stub-Prozeduren
- Server-Stub-Prozeduren
- Weitere Server-Komponenten
- ...

The diagram illustrates the compilation process. At the top is a document icon labeled 'Service Beschreibung'. An arrow labeled 'IDL-Compiler' points down to two document icons: 'Server-Stub .c' and 'Client-Stub .c'. Below these, another set of arrows labeled 'Compiler (z.B. C)' points to two more document icons: 'Server' and 'Client'. There are also two additional '.c' icons on the left and right sides, with arrows pointing towards the 'Server' and 'Client' components respectively.

Verteilte Systeme, Wintersemester 2000/2001 Folie 5.17

Client-Stub

```

Ergebnis
f_Client ( Parameter ) {
    static Server_Adresse sa;
    Auftragsnachricht an;
    Ergebnisnachricht en;

    if (sa == 0)
        sa = locate_server();

    an = Verpacken(f, Parameter);
    send(sa, an);
    receive(en);
    return Entpacken(en);
}
    
```

Wesentliche Aufgaben

- Auftragsnachricht zusammensetzen (Marshaling)
Servicenamen (Prozedurname)
Parameter
- Ergebnis zurückgeben

Heterogene (Offene) Systeme

- Konvertierung der Parameter in einen Netzstandard

Lokalisierung des Servers

- Explizite Angabe: Transparenz zum lokalen Prozeduraufruf verletzt
- Implizit: ?

Verteilte Systeme, Wintersemester 2000/2001 Folie 5.18

Die Server-Stub

... Der umgekehrte Weg:

```
f_Server ( Client ca, Auftragsnachricht an ) {
    static Server_Adresse sa;
    Ergebnismnachricht en;
    Parameter p;
    Ergebnis e;

    p := Entpacken(an);
    e := P(p);
    en := Verpacken(e);
    send(ca, en);
}
```

RPC-Server

1 Service pro Prozeß ineffizient

Zusammenfassen mehrerer
Services zu einem Server

Serverschleife

```
Serverschleife () {
    Client_Adresse ca;
    char buffer[max_auftrag];
    enum Services mt;

    while (1) {
        receive(*ca,buffer);
        mt = Servicetyp(buffer);
        switch (mt) {
            case SERVICE_f: f_Server(ca,buffer);
            break;
            ...
        }
    }
}
```

Probleme

Ideal: RPC = LPC

LPC-Semantik wird nur unvollständig erreicht

- Konzeptionelle Probleme
- Verletzung der Transparenz
- Behandlung von Ausfällen
- Performance-Probleme

Siehe auch [Tanenbaum und Renesse 1988]

RPC-Probleme (1)

Who is Who?

- Server sind passiv, Clients aktiv!
- Beispiel: Datei-Server:

$$p1 < f1 \mid p2 \mid p3 > f2$$

Wechsel in der Client/Server-Rolle

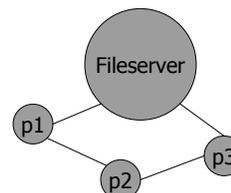
- Server hat z.B. wichtige Information, aber Client erwartet keine Nachricht vom Server

Anzahl und Typen der Parameter

- Nur bei typstrengen Sprachen problemlos
- Beispiel `printf()`

Parameterübergabe

- Call-by-Value
- Call-by-Reference?
- Call-by-Copy/Restore



RPC-Probleme (2)

Abstürze und Nachrichtenverluste gibt es bei LPC nicht

- Exceptionmechanismus (Transparenzverletzung)
- Aufrufsemantik (Exactly-Once ideal, geht aber nicht)
- Zustandsverluste bei Server/Client
- Waisenkinder (Orphans)
„Zustandsmüll“ abgestürzter Clients beim Server

Einschränkungen in der Parallelität

- ASRI komplexer

Kein Streaming möglich

LPC ist Strukturierungsmittel

- Nicht jede lokale Prozedur würde man als „Remote Procedure“ realisieren
- „Conceptual Mismatch“

Beispiel: SUN-RPC

RPC-Compiler (rpcgen) erzeugt

- Datenkonvertierungsroutinen (xdr)
- Client-Stub
- Server-Stub
- Serverschleife
- optional Skelette für Client und Server

Unterstützt UDP- und TCP-Kommunikation

Rollenwechsel umständlich aber möglich

Schutz realisierbar

Entfernte Addition

Prozedur `i_result ADD (i_param)`

RPC-Schnittstellenbeschreibung:

```

struct i_result {
    int x;
};

struct i_param {
    int a1;
    int a2;
};

program MY_RPC_SERVER {
    version MY_VERSION_1 {
        i_result ADD ( i_param ) = 1;
    } = 1;
} = 21111111;

```



Generierter Header

```

#include <rpc/rpc.h>

struct i_result {
    int x;
};

extern "C" bool_t xdr_i_result(XDR *, i_result*);

struct i_param {
    int a1;
    int a2;
};

extern "C" bool_t xdr_i_param(XDR *, i_param*);

#define MY_RPC_SERVER ((u_long)21111111)
#define MY_VERSION_1 ((u_long)1)

#define ADD ((u_long)1)
extern "C" i_result * add_1(i_param *, CLIENT *);
extern "C" i_result * add_1_svc(i_param *, struct svc_req *);

```

*Leicht
gekürzt*

Client-Stub

```

#include <memory.h> /* for memset */
#include "remote_add.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

i_result *
add_1(i_param *argp, CLIENT *clnt)
{
    static i_result clnt_res;

    memset((char *)&clnt_res, 0, sizeof(clnt_res));
    if (clnt_call(clnt, ADD, (xdrproc_t) xdr_i_param, argp,
                 (xdrproc_t) xdr_i_result, &clnt_res,
                 TIMEOUT) != RPC_SUCCESS) {
        return (NULL);
    }
    return (&clnt_res);
}

```

Verteilte Systeme, Wintersemester 2000/2001

Folie 5.27

Die Server-Stub (Eine für alle)

Fehlerbehandlung entfernt

```

static void
my_rpc_server_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
    union { i_param add_1_arg; } argument;
    char *result;
    xdrproc_t xdr_argument, xdr_result;
    char *(*local)(char *, struct svc_req *);

    switch (rqstp->rq_proc) {
    ...
    case ADD:
        xdr_argument = (xdrproc_t) xdr_i_param;
        xdr_result = (xdrproc_t) xdr_i_result;
        local = (char *(*)(char *, struct svc_req *)) add_1_svc;
        break;

    ...
    }
    (void) memset((char *)&argument, 0, sizeof (argument));
    svc_getargs(transp, xdr_argument, (caddr_t) &argument);
    result = (*local)((char *)&argument, rqstp);
    svc_sendreply(transp, xdr_result, result);
    svc_freeargs(transp, xdr_argument, (caddr_t) &argument);
}

```

Verteilte Systeme, Wintersemester 2000/2001

Folie 5.28

Die Server-Schleife

```
int main(int argc, char **argv)
{
    register SVCXPRT *transp;

    (void) pmap_unset(MY_RPC_SERVER, MY_VERSION_1);

    transp = svcdp_create(RPC_ANYSOCK);
    svc_register(transp, MY_RPC_SERVER,
                MY_VERSION_1, my_rpc_server_1, IPPROTO_UDP);

    transp = svctcp_create(RPC_ANYSOCK, 0, 0);
    svc_register(transp, MY_RPC_SERVER,
                MY_VERSION_1, my_rpc_server_1, IPPROTO_TCP);

    svc_run();
}
```

Fehlerbehandlung entfernt

Die Implementierung der entfernten Prozedur

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

i_result *add_1_svc ( i_param *p, struct svc_req *sr ) {
    static i_result result;

    result.x = p->a1 + p->a2;
    return &result;
}
```

Ein Beispiel-Client

```

#include "header.h"

void my_rpc_server_1( char* host ) {
    CLIENT *clnt;
    i_result *result_1;
    i_param add_1_arg;
    clnt = clnt_create(host, MY_RPC_SERVER, MY_VERSION_1, "udp");
    add_1_arg.a1 = 12;
    add_1_arg.a2 = 17;
    result_1 = add_1(&add_1_arg, clnt);
    printf(„Die Summe ist %d\n“,result_1->x);
    clnt_destroy( clnt );
}

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

    if(argc < 2) {
        printf("usage: %s server_host\n", argv[0]);
        exit(1);
    }
    host = argv[1];
    my_rpc_server_1( host );
}
    
```

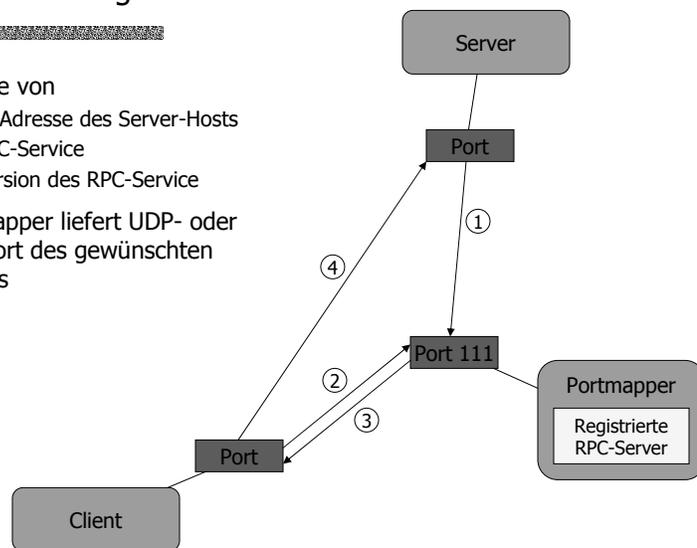
Verteilte Systeme, Wintersemester 2000/2001

Folie 5.31

Server-Lokalisierung

- Angabe von
- IP-Adresse des Server-Hosts
 - RPC-Service
 - Version des RPC-Service

Portmapper liefert UDP- oder TCP-Port des gewünschten Servers



Verteilte Systeme, Wintersemester 2000/2001

Folie 5.32

Interaktive Abfrage des Portmappers

```
rpcinfo -p balvenie.uni-trier.de
```

program	vers	proto	port	
100000	2	tcp	111	portmapper
100000	2	udp	111	portmapper
100004	2	udp	789	ypserv
100004	1	udp	789	ypserv
100004	2	tcp	792	ypserv
100009	1	udp	790	yppasswdd
100007	2	udp	792	ypbind
100007	2	tcp	794	ypbind
100003	2	udp	2049	nfs
100003	2	tcp	2049	nfs
100005	1	udp	829	mountd
100005	2	udp	829	mountd
100005	1	tcp	832	mountd
100005	2	tcp	832	mountd

Verteilte Systeme, Wintersemester 2000/2001

Folie 5.33

Übungsaufgaben

1. Auf dem Server balvenie.uni-trier.de existiert ein RPC-Server mit folgender Schnittstellenbeschreibung:

```
struct my_arg {
    int v;
};

struct my_result {
    char v[80];
};

program SOME_SERVER {
    version SOME_SERVER_V1 {
        my_result QUERY ( my_arg ) = 42;
    } = 1;
} = 21111112;
```

- Implementieren Sie einen RPC-Client, der obigen Server ein- oder mehrfach anfragt. Versuchen Sie, das Resultat zu interpretieren.

Verteilte Systeme, Wintersemester 2000/2001

Folie 5.34

Literatur

A.D. Birrell, B.J. Nelson
Implementing Remote Procedure Call
ACM Transactions on Computer Systems
Vol. 2, No. 2, pp. 39-59, 1984

A.S. Tanenbaum, R. van Renesse
A Critique of the Remote Procedure Call Paradigm
in: Research into Networks and Distributed Applications
R. Speth (Hrsg.), Elsevier Science Publishers, pp. 775-783, 1988