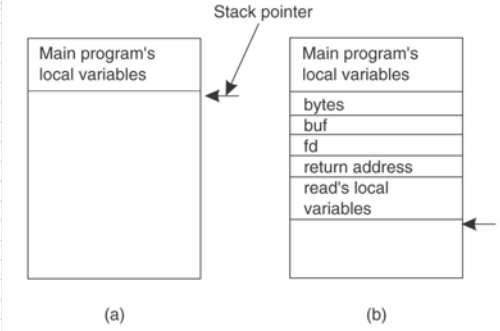
RPC – Remote Procedure Call

Apel de procedura conventional

- count = read (fd, buf, bytes)
  - Executia
    - programul apeleaza o rutina de biblioteca
    - aceasta face un apel sistem (OS)
    - OS umple buf
    - si intoarce rezultatul
  - Parametri
    - apel prin valoare
    - apel prin referinta
    - apel prin "copy / restore"

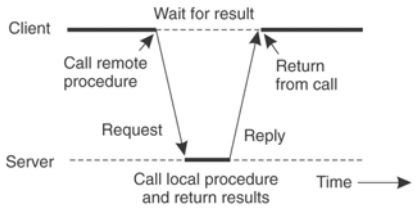
Comunicarea parametrilor



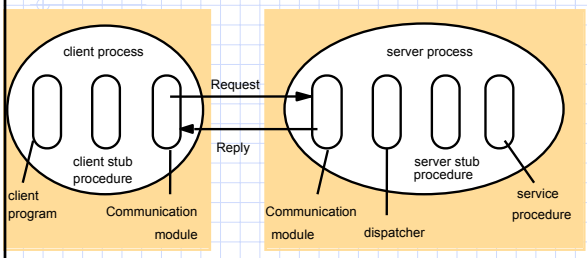
Principiul RPC (apel sincron)

Concept introdus de Birrell si Nelson in 1984:

un proces pe masina A apeleaza o procedura pe masina B
procesul este suspendat si executia continua pe B
rezultatul procedurii este intors procesului
la primirea lui, procesul continua executia



Rolul procedurilor stub la client si server



Stub-uri client si server

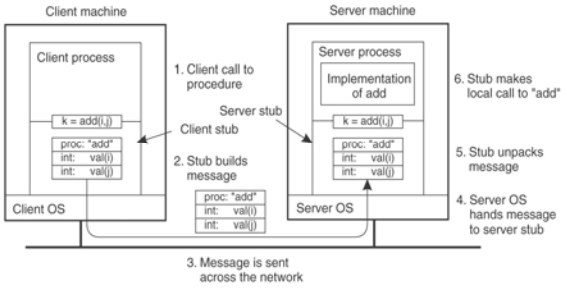
- apel RPC
  - client
    - clientul apeleaza stub-ul client
    - stub-ul impacheteaza parametrii pentru procedura la distanta (conform unui format standard)
    - stub-ul apeleaza OS (modulul comunicare) sa transmita mesajul
    - asteapta raspunsul
  - server
    - OS apeleaza un stub server care despacheteaza parametrii de la formatul standard la cel al masinii server
    - stub-ul server apeleaza procedura server

Stub-uri client si server (continuare)

- serverul umple buf (intern stub-ului)
- stub-ul server impacheteaza rezultatul si transmite mesajul stub-ului client

- client
  - OS da mesajul stub-ului client
  - stub-ul client despacheteaza, copiaza datele in buf(-ferul) client si intoarce rezultatul

Executia unui calcul la distanta cu RPC



Probleme: Reprezentarea datelor

(a) Mesajul original pe Pentium (b) Mesajul receptionat pe SPARC (c) Mesajul dupa inversare

Transferul parametrilor

- Marshalling
  - parametri valoare
    - coduri diferite
    - little endian, big endian
  - parametri referinta
    - mult mai dificil
    - merge pentru structuri simple (arrays)
    - copy / restore merge in anumite cazuri
    - pentru eficienta comunicatiei: in, out, in / out
    - structuri complexe (grafuri)
      - linearizarea structurii, transmiterea si refacerea ei la celalalt capat

O procedura si mesajul corespunzator

```
foobar( char x; float y; int z[5] )
{
  ....
}
```

(a)

foobar's local variables
x
y
5
z[0]
z[1]
z[2]
z[3]
z[4]

(b)

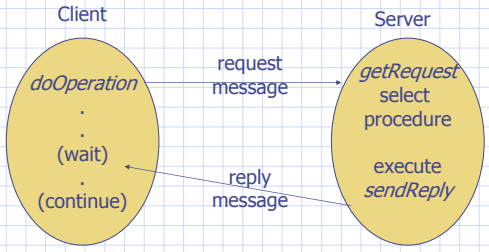
Specificarea parametrilor

- Protocol RPC se refera la
  - formatul datelor
    - reprezentatare tipuri si structuri de date
      - folosesc tipuri specifice (XDR -eXternal Data Representation)
      - sau generale (ASN.1—Abstract Syntax Notation)
    - Interface Definition Language
      - specificare compilata in stub-uri client si server
  - regulile schimbului de mesaje (de ex. TCP)

Protocoale RPC – schimbul de mesaje

Protocol	Client	Server	Client
R	request		
RR	request	reply	
RRA	request	reply	ack

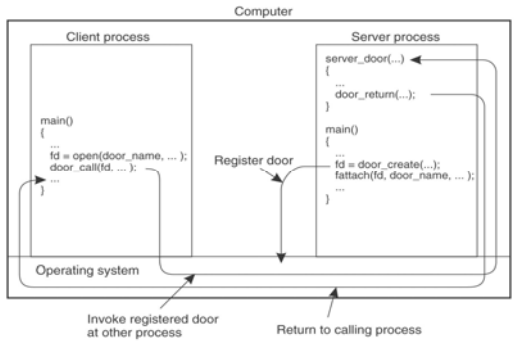
Comunicarea cerere - raspuns



Semantica invocarilor RPC

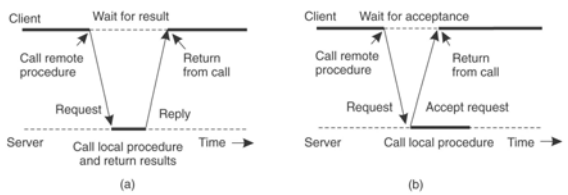
retransmite *request*	Filtrare duplicate	Re-executie procedura sau retransmite *reply*	Semantica invocarii
Nu	Ne-aplicabil	Ne-aplicabil	*Maybe*
Da	Nu	Re-executa procedura	*At-least-once*
Da	Da	Retransmite *reply*	*At-most-once*

Porti (doors) ca mecanisme IPC

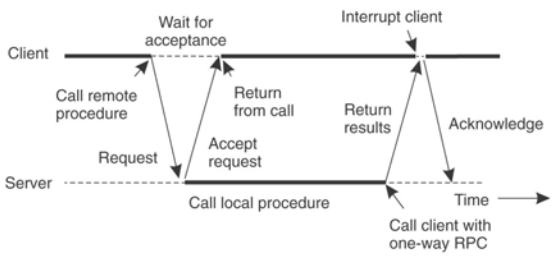


RPC sincron si asincron

(a) RPC traditional (b) RPC asincron



RPC dublu asincron



SUN RPC

Numit si **ONC RPC (ONC – Open Network Computing)**
Utilizat in scrierea **NFS**
Poate fi folosit peste **TCP** sau **UDP** la alegere
Are un **limbaj de descriere a interfetei, XDR**
Ofera un **compilator asociat, rpcgen**

XDR – eXternal Data Representation standard

Specificat in **RFC 1014**
Permite definirea
Procedurilor utilizabile de la distanta
Tipurilor asociate:
Constante, typedefs, structuri, uniuni, enumerari

Caracteristici
- Interfata identificata prin **nr program** si **nr versiune**
- O definitie de procedura include **semnatura si nr procedura**
- Semnatura procedurii include
  tip_rezultat nume_procedura tip_parametru_intrare
- Un singur parmetru de intrare si un singur rezultat (restrictie eliminata)

Exemplu - Airline reservation RPC interface definition

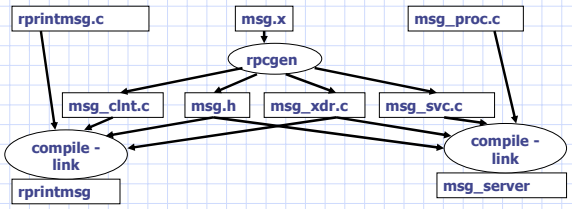
```
struct reserve_args {
  int flight_number;
  string passenger_name<>;
};

struct reserve_result {
  int ok;
  string seat<>;
};

program RESERVATION_PROG {
  version RESERVATION_VERS {
    reserve_result RESERVATION_RESERVE (reserve_args) = 1;
  } = 1;
} = 400001;
```

rpcgen

genereaza urmatoarele fisiere (pe baza definitiei din fisierul msg.x):
- procedurile din **stub client** (**msg_clnt.c**)
- procedurile din server: **main, dispecer, stub** (**msg_svc.c**)
- procedurile de **marshalling** si **unmarshalling** XDR (**msg_xdr.c**)
- fisier antet corespunzator definitiilor de proceduri si tipuri din **msg.x** (**msg.h**)

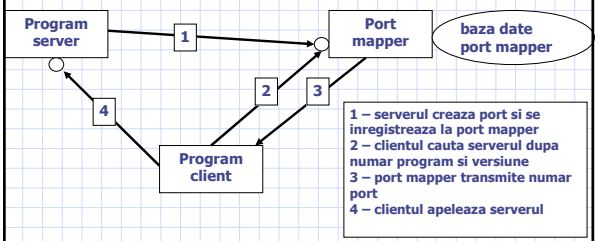


Modificari client si server

- Clientul
  - initializeaza interfata RPC cu *clnt_create*
  - aceasta creaza un **RPC handle** la programul si versiunea specificate, pe o gazda data
  - apelata inaintea oricarei proceduri la distanta
  - parametri:
    - nume server
    - nume si versiune program (generate de rpcgen in msg.h)
    - protocolul de transport utilizat
- Server
  - executa stub si pune procesul in background
  - creaza un socket si leaga un port local la socket
  - apeleaza *svc_register* pentru a inregistra portul, numarul si versiunea programului (se contacteaza *port mapper*)
  - executa *listen* (asteapta cererile clientilor)

Binding

Foloseste **port mapper** aflat la un port cunoscut pe fiecare masina care ruleaza RPC. La el se inregistreaza toate **serverele** de pe masina locala prin: **nr program, nr versiune, nr port**



Exemplu

date.x

```
/* date.x - description of remote date service */
/* we define two procedures: */
/* bin_date_1 returns the time in binary format */
/* (seconds since Jan 1, 1970 00:00:00 GMT) */
/* str_date_1 converts a binary time to a readable date string */

program DATE_PROG {
version DATE_VERS {
long BIN_DATE(void) = 1; /* procedure number = 1 */
string STR_DATE(long) = 2; /* procedure number = 2 */
} = 1;
} = 0x31415926;
```

server.c

```
#include <rpc/rpc.h>
#include "date.h"
/* bin_date_1 returns the system time in binary format */
/* name obtained from BIN_DATE with lower case and version number 1*/
/* result is a pointer to the value declared in the RPC definition */
long * bin_date_1()
{
static long timeval;      /* must be static!! This value is */
/* used by rpc after bin_date_1 returns */

long time();              /* Unix time function; returns time */
timeval = time((long *)0);
return &timeval;
}
/* str_date_1 converts a binary time into a date string */
char ** str_date_1(bintime)
long *bintime;
{
static char *ptr;         /* return value... MUST be static! */
char *ctime();            /* Unix library function that does the work */
ptr = ctime(bintime);
return &ptr;
}
```

client.c

```
/* client code */
#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>
#include "date.h"
main(argc, argv)
int argc;
char **argv;
{
CLIENT *cl;               /* rpc handle */
char *server;
long *lresult;            /* return from bin_date_1 */
char **sresult;           /* return from str_date_1 */
if (argc != 2) {
fprintf(stderr, "usage: %s hostname\n", argv[0]);
exit(1);
}
server = argv[1];         /* get the name of the server */
```

```
/* create the client handle */
if ((cl=clnt_create(server, DATE_PROG, DATE_VERS, "netpath")) ==NULL)
/* netpath - search a NETPATH environment var for a sequence of preferred
    transport providers */
{   /* failed! */
    clnt_pcreateerror(server);
    exit(1);
}
/* call the procedure bin_date */
if ((lresult=bin_date_1(NULL, cl))==NULL) {
    /* failed ! */
    clnt_perror(cl, server);
    exit(1);
}
printf("time on %s is %ld\n", server, *lresult);
/* have the server convert the result to a date string */
if ((sresult=str_date_1(lresult, cl)) == NULL) {
    /* failed ! */
    clnt_perror(cl, server);
    exit(1);
}
printf("date is %s\n", *sresult);
clnt_destroy(cl);          /* get rid of the handle */
exit(0);
```

Suport de securitate

Trei forme de autentificare

AUTH_NONE	fara autentificare
AUTH_UNIX	autentificare prin uid si gid
AUTH_DES	autentificare prin credential de forma user@machine;

verificatorul este un marcaj de timp criptat DES

Forma mesajelor

Call	**Reply**
Xid	Xid
Call / reply	Call / reply
Rpc version	Accepted? (vs bad rpc version or auth failure)
Program #	Auth stuff
Program version	Success? (vs bad prog / proc #)
Procedure #	results
Auth stuff	
Arguments	

Reprezentarea datelor pentru XDR

int	complement fata de 2, 32 biti, little endian
unsigned int	32 biti little endian
float	32 biti: semn, exponent pe 8 biti, mantisa pe 23 biti
double	64 biti: semn, exponent pe 11 biti, mantisa pe 52 biti
string	lungime sir pe 4 octeti plus (n+r) octeti pentru sir umplut la multiplu de 4 octeti
tablou	lungime fixa: n elemente de la 0 la n-1
	lungime variabila: lungime pe 4 octeti urmata de n elemente de la 0 la n-1
struct	componentele in ordinea declararii
union	discriminant pe 4 octeti urmat de valoarea de pe varianta selectata
void	0 octeti

DCE RPC

- Dezvoltat de OSF (acum Open Group)
- Proiectat pentru Unix
- Extins la alte OS (VMS, Windows NT)
- Bazat pe modelul client-server
- O parte din serviciile client-server sunt incluse in DCE RPC
  - Distributed file service
  - Directory service
  - Security service
  - Distributed time service

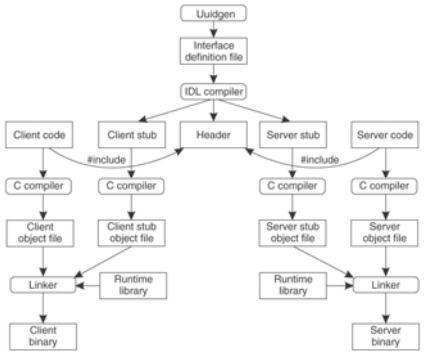
Obiective DCE RPC

- Transparenta locatiei
- Transport mesaje intre client si server
- Conversie tip date
- Transparenta limbajului (ex. client Java, server C)
- Transparenta OS
- Transparenta hardware
- Transparenta protocoalelor de retea

Caracteristici

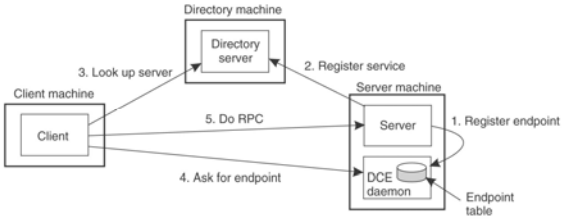
- Similar Sun RPC
  - Interfete scrise in *Interface Definition Notation (IDN)*
  - Un compilator IDN genereaza header-ul si stub-urile client si server
- Diferente
  - identificare server
    - Sun – numar unic 32 biti ales de programator
    - DCE – ID unic de 128 biti generat prin program *uuidgen* care genereaza si un fisier prototip IDN

Scriere client si server in DCE RPC



Legare client - server in DCE

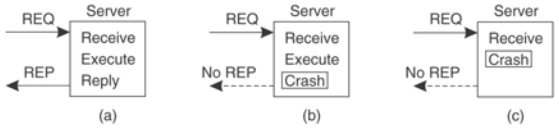
- legarea la server
  - Sun – programatorul trebuie sa cunoasca masina serverului
  - DCE – serverul inregistreaza endpoint cu RPC daemon local si corespondenta {nume_server, masina} cu un server de directoare



Semantici RPC in prezenta defectelor

- Clientul nu poate localiza serverul
  - Solutii
    - Provoaca exceptie sau foloseste handler de semnalizare (in C)
  - Neajunsuri
    - nu orice limbaj are exceptii sau handler de semnalizare
    - distruge transparenta
- Mesaj de cerere pierdut
  - Solutie
    - Timer in OS client
  - Neajuns
    - Merge daca mesajul a fost cu adevarat pierdut
    - Altfel, serverul trebuie s faca diferenta intre original si copie

Caderea Serverului



a – normal
b – crash dupa executie
c – crash inainte de executie

cand apare un timeout, clientul nu poate diferentia intre b si c

solutii pentru client
- repeta cererea pana reuseste (at least once)
- semnaleaza eroarea (at most once)
- nimic (maybe)
- solutie teoretica? (exactly once)

Exactly once - Strategii client si server

- Client-server pentru tiparire text
- Strategii server
  - trimite ACK inainte de a spune printer sa tipareasca
  - trimite ACK dupa tiparire text
- *Server anunta ca a cazut si acum este iar operational*
- Strategie retransmitere cerere client
  - mereu
  - niciodata
  - doar cand nu s-a confirmat trimiterea cererii la server
  - doar cand s-a confirmat cererea de tiparire

Combinatii

- Evenimente
  - transmite mesaj terminare (M)
  - tipareste text (P)
  - cadere server (C)
- Combinatii
  - M P C
  - M C (P)
  - P M C
  - P C (M)
  - C (P M)
  - C (M P)

Combinatii (2)

Client	Server						
	Strategy M → P			Strategy P → M		
Reissue strategy	MPC	MC(P)	C(MP)	PMC	PC(M)	C(PM)
Always	DUP	OK	OK	DUP	DUP	OK
Never	OK	ZERO	ZERO	OK	OK	ZERO
Only when ACKed	DUP	OK	ZERO	DUP	OK	ZERO
Only when not ACKed	OK	ZERO	OK	OK	DUP	OK

OK = Text is printed once
DUP = Text is printed twice
ZERO = Text is not printed at all

Pierdere mesaje reply

- Solutie – timer
- Dezavantaje
  - merge la operatii idempotente
  - pentru celelalte
    - **clientii asociaza cererilor numere de secventa si serverul tine evidenta lor**
    - **un bit in mesaj distinge originalul de copii**

Cadere Client

- Calcule orfane
- Probleme
  - consum inutil resurse
  - fisiere incuiate (lock)
  - sincronizare cu client re-boot-at
- Solutii
  - exterminare (log entry)
  - reincarnare (epoci)
  - reincarnare lina (localizare proprietar)
  - expirare (bucata de timp T standard)

Concluzii

- Conceptul RPC a fost extins la obiecte (DCOM, CORBA, Java RMI)
- Aspectele de semantica in caz de defectare raman valide in acest context