




**Construction d'application avec
RPC**

Introduction

- Souvent, la communication par socket = invocation de comm à distance.
- Problèmes :
 - Lourd à programmer : Encodage des données (paramètres, résultats, ...), identification du serveur, du protocole, etc.
 - Pas naturel.
 - Élaboration d'un énorme *switch* au niveau du serveur.
- Retrouver la sémantique habituelle de l'appel de procédure :
 - sans se préoccuper de la localisation de la procédure,
 - sans se préoccuper du traitement des défaillances.

Introduction



- Les difficultés :

- Appel de procédures locales :

- Appelant et appelé dans le même espace virtuel : même mode de pannes, appel et retour fiable.

- Appel de procédures distantes :

- Appelant et appelé dans deux espaces virtuels : mode de pannes indépendant, réseau non fiable, temps de réponse.



Introduction

Besoin d'un environnement de haut niveau pour le développement d'applications réparties qui :
reprend le concept du client/serveur

permet d'identifier un très grand nombre de services (> #n° de port)

conserve les paradigmes habituels d'exécution :

- l'appel de fonction, passage de paramètres
- la notion de programme (ensemble de fonctions)

☞ **RPC**

masque l'hétérogénéité de représentation des données

- format standard, fonctions de transcodage

☞ **XDR**

description des structures de données. langage de description : RPC language

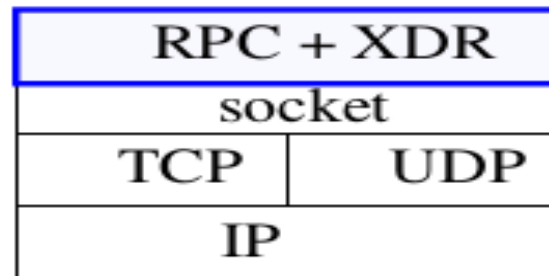
accompagné d'un outil de génération automatique :

☞ **RPCgen**

Introduction

environnement de plus haut niveau que les “sockets” et la transmission de messages.

⇒ Architecture fonctionnelle sur Internet :



TI-RPC = Transport independent RPC

- Implémentation des RPC qui permet le développement d'applications indépendamment des éléments logiques et physiques (réseaux, protocoles, etc.) utilisés pour transmettre des données.



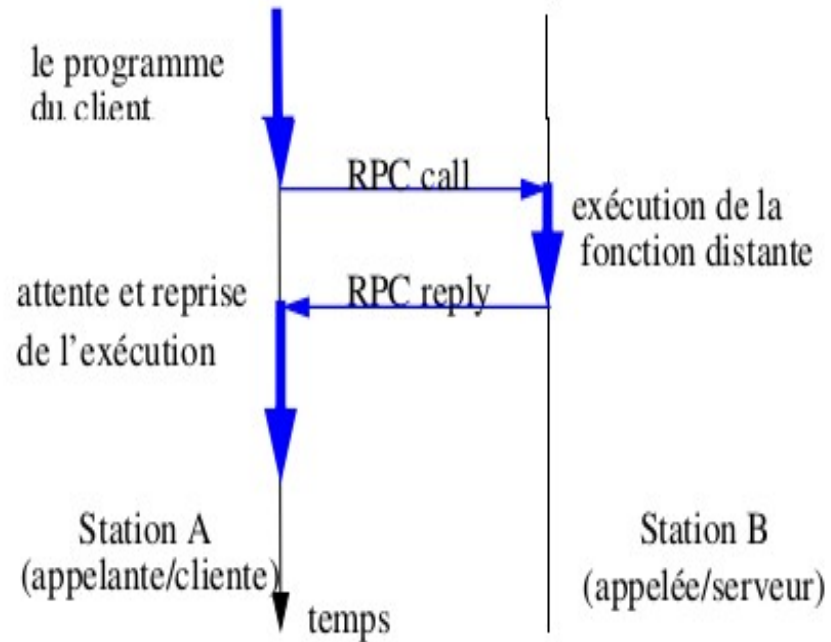
RPC : Objectifs

Objectifs :

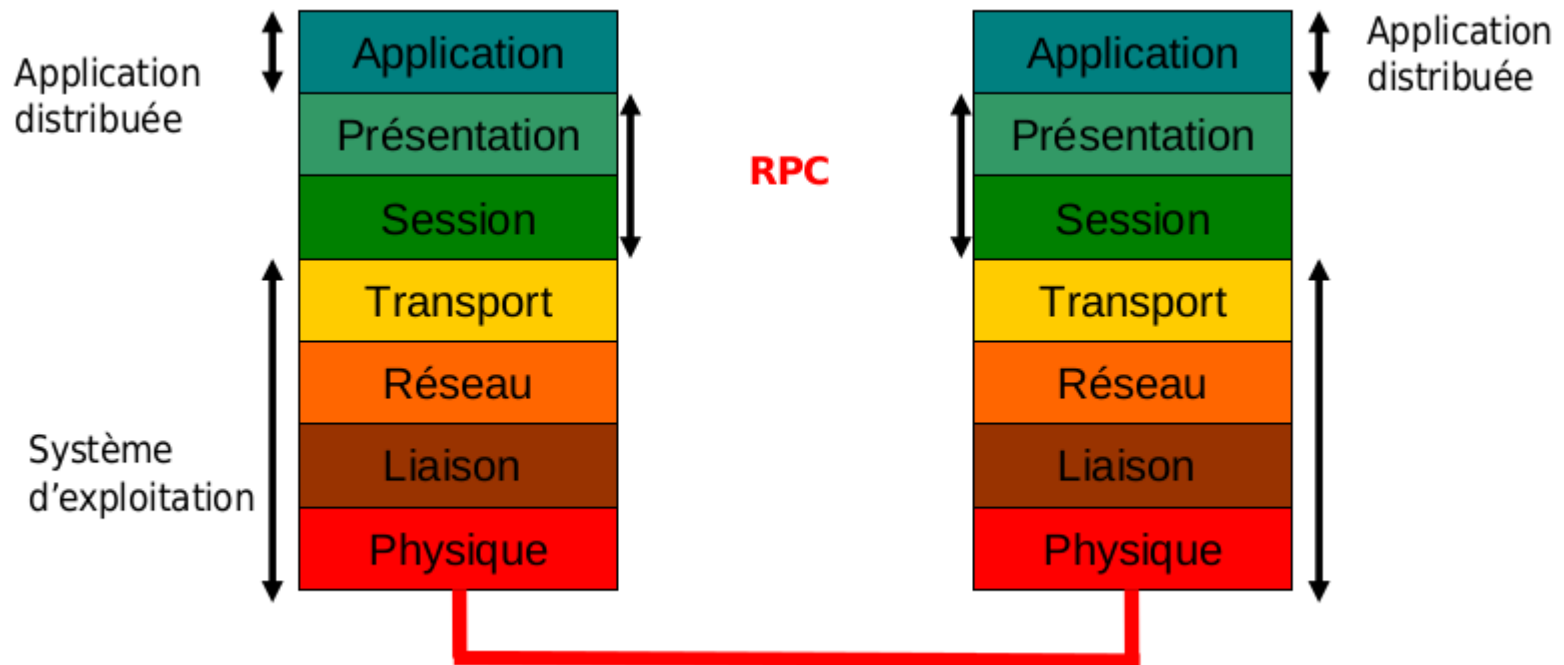
- Model de programmation (classique) basé sur l'appel de procédures au lieu du mécanisme L'envoi/réception de messages
- Cacher la complexité du réseau au développeurs d'applications distribuées (Sockets, numéro de port, Formatage des donnée, ordre des octets,...)

Présentation

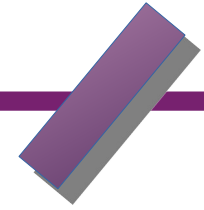
Exécution d'une procédure à distance :



RPC dans le modèle OSI



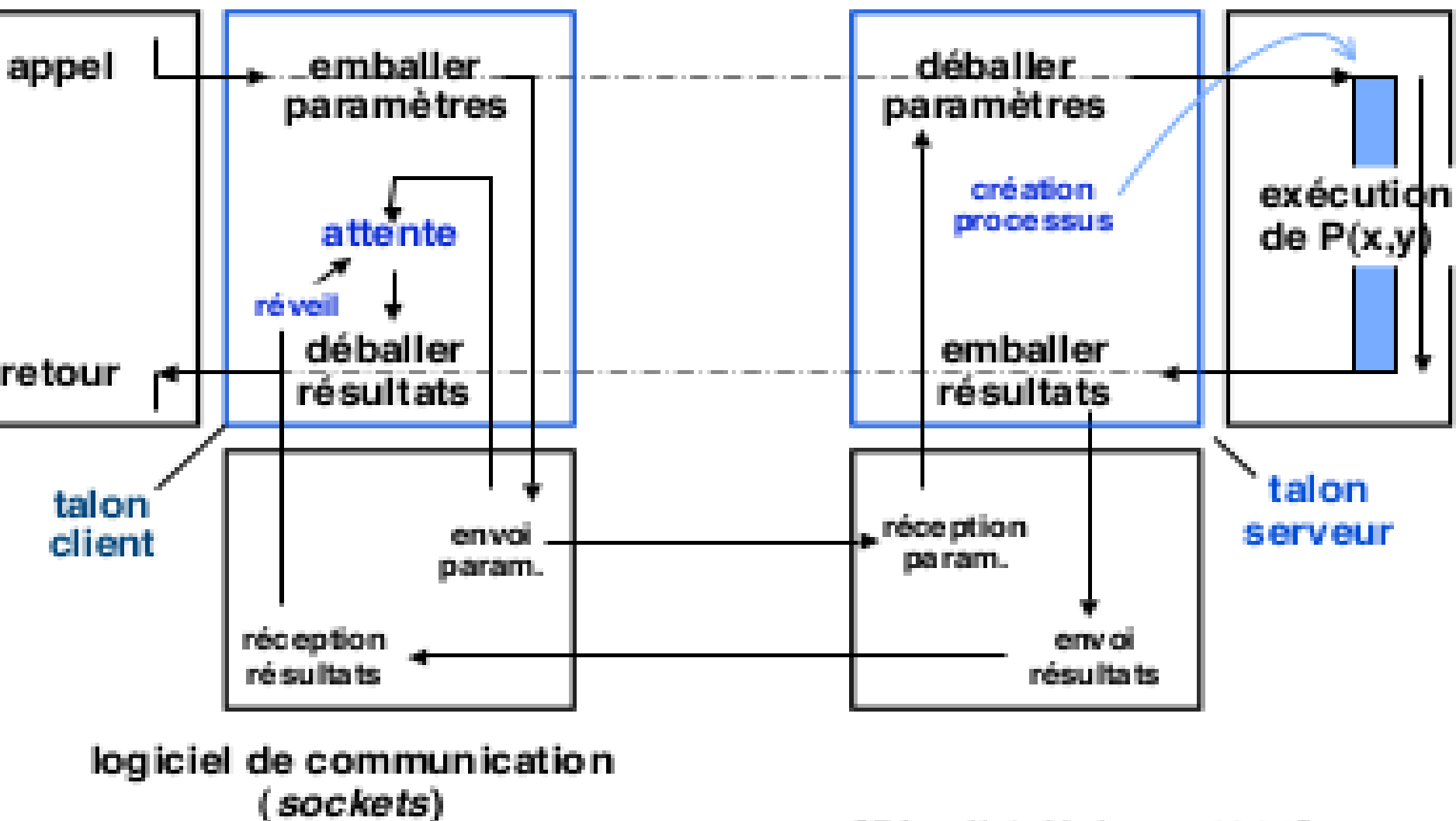
Le RPC



La sémantique de l'appel :

- Si on tient compte des erreurs (**pertes, duplications**) pouvant survenir lors des communications, on définit 3 sémantiques possibles pour l'appel de procédures distantes :
 - exactement une fois
 - au moins une fois
 - au plus une fois.
- La sémantique choisie par l'implémentation sous RPC-Sun est **au moins une fois**
 - de ce fait il faut s'assurer que l'exécution d'une procédure distante soit idempotent,
 - . par exemple en utilisant **le numéro de transaction (xid)** disponible dans chaque message RPC.

Le RPC : principe de réalisation

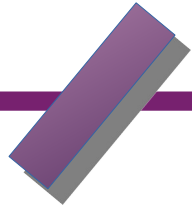


Fonctions des talons (stubs)

Talon client

- Représente le serveur sur le site client
 - Reçoit l'appel local
 - Emballe les paramètres
 - Crée un identificateur unique pour l'appel
 - Exécute l'appel
 - Met le proc. client en attente
 - Reçoit et déballe les résultats
- !

Fonctions des talons (stubs)



Talon serveur

- Représente le client sur le site serveur
 - Reçoit l'appel sous forme de message et
 - déballe les Crée ou sélectionne un processus (ou thread) et lui fait exécuter la procédure
- Emballer les résultats et les transmet à l'appelant
- " Exécute le retour vers l'appelant (comme retour local de procédure)

Le RPC

Les paramètres

- **un seul** paramètre est échangé lors de l'appel (RPC call)
 - si l'application requiert l'échange de plusieurs paramètres ils doivent être regroupés au sein d'une seule structure de données.
- **un seul** élément peut être échangé lors du retour (RPC reply)
 - à travers la valeur de retour de la fonction

Le RPC

Identification des procédures distantes

- Une procédure distante est identifiée de manière unique par un triplet :

- #program. #proc version. #procedure

- U

Tableau 1 : les numéros de programme

une

vers

- pl

- C

numéro de programme	utilisation
0000.0000 - 1FFF.FFFF ₁₆	pour des services généraux
2000.0000-3FFF.FFFF ₁₆	pour des services en cours de développement
4000.0000-5FFF.FFFF ₁₆	attribués dynamiquement
6000.0000-FFFF.FFFF ₁₆	réservés

ices :



RPC : Principes

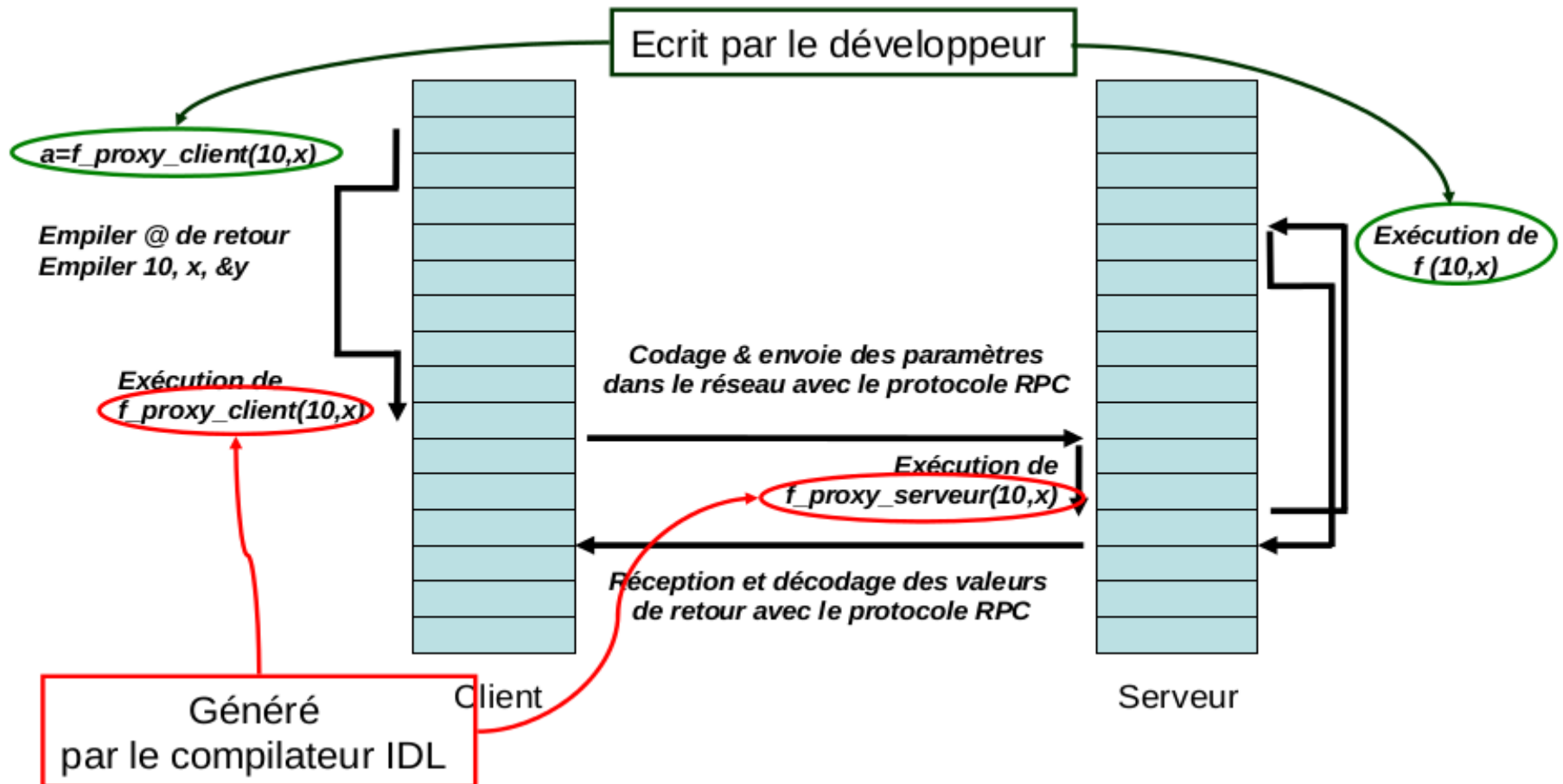
RPC Gère :

- L'enregistrement/localisation des procédures dans un réseau
- Codage des données et transfert de paramètres dans le réseau

Le développeur :

- Écrit le code de la procédure en un langage de haut niveau (C, C++, Java, ...)
- Définit l'interface de la procédure en utilisant IDL
- Génère le code source des proxys (client & serveur) à l'aide d'un compilateur IDL

RPC : Principes



IDL : Interface Definition Language

Langage de haut niveau pour définir l'interface d'une procédure

S'intéresse à définir les paramètres d'entrée et de sortie d'une procédure.

(Ne définit pas l'implémentation interne)

Utilisé par le compilateur **rpcgen** pour générer le code des proxy client et

serveur

IDL : Interface Definition Language

Définition de nouveaux types

```
program NOM_PROGRAMME {  
version VERSION_PROG {  
type PROCEDURE1 (type) = id_procedure;  
type PROCEDURE2 (type) = id_procedure;  
...  
}= id_program
```

IDL : Interface Definition Language

- Chaque programme est défini par **nom** et un **identificateur**
 - Un programme comporte **plusieurs procédures**
 - Chaque procédure est définie par **un nom**, **type d'entrée**, **type de retour** et un **identificateur**
 - Un programme peut être écrit dans **plusieurs version**
- Les fichier IDL ont l'extension **.X**



Compilateur IDL : rpcgen

Un compilateur IDL **rpcgen** est un programme qui à partir d'un code écrit en IDL génère le code en C :

- Du proxy client
- Du proxy serveur
- Des procédures de conversion de données
- Des fichiers entêtes

rpcgen peut également générer des squelettes du programme client et de la procédure distante

IDL : Interface Definition Language

```
program PROG {  
version PROG_V1 {  
long FACT(int) = 1;  
} = 1;  
} = 0x20000001;
```

Exemple : programme factorielle **fact.X**

IDL : Interface Definition Language

Type de données IDL : Semblables au langage C

- `const MAX_SIZE=128`
- **Types de base:** char, short, int, long, float.
 - `int x; float f;`
- **Chaine de caractères**
 - `string nom<64>;`
- **Tableau**
 - `Int tab[128];`
- **Type structuré**
 - `struct point {int x,y};`
- **Création de nouveau types**
 - `typedef int tableau [MAX_SIZE];`

IDL : Interface Definition Language

Règles de génération de code C : Pour chaque procédure définie dans le fichier IDL, rpcgen une procédure

- Qui porte le même nom avec la définition IDL
 - En miniscule
 - Suivie de `_versionprog_svc` : pour la procédure proxy serveur
 - Suivie de `_versionprog` : pour la procédure proxy client

Ex FACT : `fact_1 & fact_1_svc`

- Dont les arguments de la procédure sont des pointeur vers les type d'arguments spécifié dans IDL
- Qui retourne un pointeur vers le type spécifié dans IDL. Ce pointeur doit être vers une variable **static**

Procédures avec plusieurs paramètre d'entrées :

- RPC support des procédure avec un seul paramètre
- Il faut utiliser un structure pour passer plusieurs paramètres

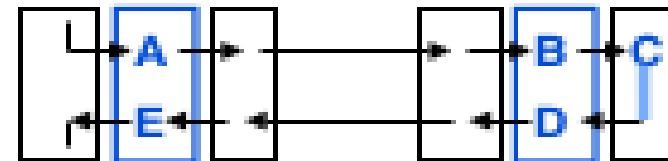
RPC : “sémantique” en cas de panne



Hypothèses de défaillances

- **Systeme de communication**
 - Congestion du réseau, messages retardés
 - Perte de messages
 - Altération de messages
- **Serveur**
 - Défaillance avant l'exécution de la procédure
 - Défaillance pendant l'exécution de la procédure
 - Défaillance après l'exécution de la procédure
 - Défaillance définitive ou reprise possible
- **Client**
 - Défaillance pendant le traitement de la requête

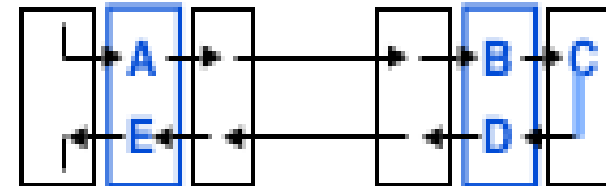
RPC : traitement des défaillances



■ Congestion du réseau ou du serveur

- ◆ Panne transitoire (ne nécessite pas d'action)
- ◆ Détection : expiration du délai de garde A ou D
- ◆ Reprise (délai de garde A)
 - ❖ Le talon client (A) réémet l'appel (même identificateur) sans intervention de l'application
 - ❖ Le service d'exécution (C) détecte que c'est une réémission
 - ▲ Appel en cours : aucun effet
 - ▲ Retour déjà effectué : réémission du résultat
- ◆ Reprise (délai de garde D) : réémission du résultat
- ◆ Sémantique
 - ❖ Si défaillance transitoire : exactement une fois
 - ❖ Si défaillance permanente : détection, exception vers application

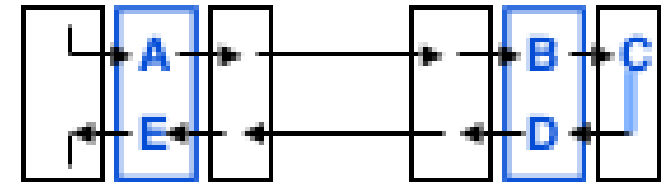
RPC : traitement des défaillances



■ Panne du serveur après émission de l'appel

- ◆ Plusieurs moments possibles
 - ❖ Avant B, entre C et D, entre fin traitement et D
 - ❖ Traitement : pas fait, partiel, total
- ◆ Détection : expiration du délai de garde A
- ◆ Reprise
 - ❖ Le client réémet l'appel dès que le serveur redémarre
 - ❖ Sémantique : **au moins une fois**
 - ▲ Le client ne connaît pas l'endroit de la panne
 - ❖ On peut faire mieux avec un service transactionnel
 - ▲ Mémorise l'identificateur de requête et état avant exécution

RPC : traitement des défaillances



■ Panne du client après émission de l'appel

- ◆ L'appel est correctement traité
 - ❖ Changement d'état du serveur
 - ❖ Le processus exécutant courant est déclaré "orphelin"
- ◆ Détection : expiration du délai de garde D, n réémissions infructueuses
 - ❖ Action du serveur : élimination des orphelins
 - ▲ Consommement des ressources
- ◆ Reprise (après redémarrage du client)
 - ❖ L'application cliente réémet l'appel (id. différent)
 - ▲ Sémantique : au moins une fois
 - ▲ Le serveur ne peut pas détecter qu'il s'agit d'une répétition
 - ◊ Pas d'incidence si idempotent
 - ▲ On peut faire mieux, si le client a un service de transactions

Exemple : factorielle

```
$cat fact.X
program PROG {
    version PROG_V1 {
        long FACT(int) = 1;
    } = 1;
} = 0x20000001;

$rpcgen fact.X
$rpcgen -Ss fact.X > remote_fact.c
$rpcgen -Sc fact.X > local_fact.c
$ls
fact_clnt.c  fact.h  fact_svc.c  fact.X  local_fact.c  remote_fact.c
$
```

Exemple : fact.h

```
#ifndef _FACT_H_RPCGEN
#define _FACT_H_RPCGEN

#include <rpc/rpc.h>

#ifdef __cplusplus
extern "C" {
#endif

#define PROG 0x20000001
#define PROG_V1 1

#if defined(__STDC__) || defined(__cplusplus)
#define FACT 1
extern long * fact_1(int *, CLIENT *);
extern long * fact_1_svc(int *, struct svc_req *);
extern int prog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

#ifdef __cplusplus
}
#endif
#endif /* !_FACT_H_RPCGEN */
```

Exemple : fact_clnt.c

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

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

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

long *
fact_1(int *argp, CLIENT *clnt)
{
    static long clnt_res;

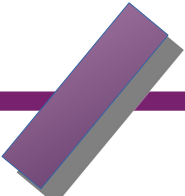
    memset((char *)&clnt_res, 0, sizeof(clnt_res));
    if (clnt_call (clnt, FACT,
                  (xdrproc_t) xdr_int, (caddr_t) argp,
                  (xdrproc_t) xdr_long, (caddr_t) &clnt_res,
                  TIMEOUT) != RPC_SUCCESS) {
        return (NULL);
    }
}
```

Exemple : local_fact.c


```
void
prog_1(char *host)
{
    CLIENT *clnt;
    long *result_1;
    int fact_1_arg=5;
#ifdef DEBUG
    clnt = clnt_create (host, PROG, PROG_V1, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */
    result_1 = fact_1(&fact_1_arg, clnt);
    if (result_1 == (long *) NULL) {
        clnt_perror (clnt, "call failed");
    }
#ifdef DEBUG
    printf("%d\n", *result_1);
    clnt_destroy (clnt);
#endif /* DEBUG */
}

int
main (int argc, char *argv[])
{
    char *host;
    if (argc < 2) {
        printf ("usage: %s server host\n", argv[0]);
    }
}
```


Execution



```
$gcc -o serveur fact_svc.c remote_fact.c
$gcc -o client fact_clnt.c local_fact.c
$ service rpcbind status
rpcbind (pid 1419) is running...
$./serveur &
[1] 2704
$./client localhost
120
$
```



RPC : Difficultés techniques



L'appel de procédures distantes présente des difficultés concernant :

- Passage de paramètres
- Localisation des procédures distantes
- Sémantiques de l'appel
- Représentation des données

RPC : Difficultés techniques

Passage de paramètres

- Pas de passage de paramètre par adresse
- La valeur des pointeurs n'est pas significative dans un ordinateur distant

RPC : Difficultés techniques

Localisation des procédures distantes

- Utilisation d'un service de résolution de nom

RPC : Difficultés techniques

Sémantiques de l'appel

- Si le client ne reçoit pas de réponse au bout d'un temps déterminé, soit
 1. Le message envoyé a été perdu
 2. La réponse du serveur a été perdu
 3. Le serveur a eu une défaillance
- Si le client envoie de nouveau la requête
 - Pas de problèmes dans le cas 1
 - La requête sera exécuté 2 fois dans le cas 2
 - Divers résultat possibles



RPC : Difficultés techniques

Représentation des données

- XDR : eXternal Data Representation