

Université de Lille1

Génération de RPC à l'aide de RPCgen

N. Melab (melab@lfl.fr)

DESS-ISIDIS

Les RPC

Modèle et fonctionnement des RPC

DESS-ISIDIS

Les RPC

Approches de conception d'applications C/S

- Conception orientée communication
 - ☒ Définition du protocole de communication (format et syntaxe des messages échangés par le client et le serveur)
 - ☒ Conception du serveur et du client en spécifiant comment ils réagissent aux messages échangés
- Conception orientée traitement
 - ☒ Construction d'une application conventionnelle dans un environnement mono-machine
 - ☒ Subdivision de l'application en plusieurs modules pouvant s'exécuter sur différentes machines

DESS-ISIDIS

Les RPC

Conception orientée communication

- Problèmes
 - ☒ Gestion des formats de messages et données par l'utilisateur (hétérogénéité)
 - ☒ Empaquetage/déempaquetage des messages
 - ☒ Le modèle est souvent asynchrone, ce qui rend la gestion des erreurs plus complexe
 - ☒ Le modèle n'est pas naturel pour la plupart des programmeurs
 - ☒ Communication explicite et non transparente

DESS-ISIDIS

Les RPC

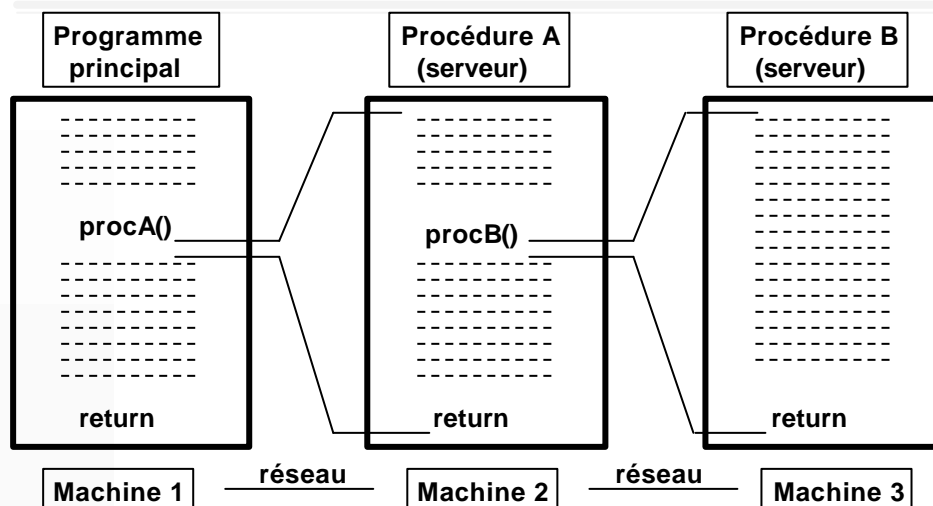
Conception orientée application

- Objectif : Garder la démarche de conception des applications centralisées
- Appel de procédure à distance ou *Remote Procedure Call (RPC)*
 - ☒ Introduit par Birrell & Nelson (1984)
 - ☒ Garder la sémantique de l'appel de procédure local ou *Local Procedure Call (LPC)*
 - ☒ Fonctionnement synchrone
 - ☒ Communication transparente entre le client et le serveur

DESS-ISIDIS

Les RPC

RPC : principe



DESS-ISIDIS

Les RPC

Le modèle

- Le modèle LPC
- Le modèle RPC

DESS-ISIDIS

Les RPC

Le modèle LPC

- Notion de contexte et de pile d'exécution
- Déroulement
 - ↳ Empilement des paramètres
 - ⇔ Copie dans la pile des paramètres passés par valeur
 - ⇔ Empilement des références des paramètres passés par adresse
 - ↳ Empilement de l'adresse de retour
 - ↳ Empilement des variables locales
 - ↳ Exécution du code de la procédure

DESS-ISIDIS

Les RPC

Passage de paramètres

- Appel par valeur
 - ☒ Copie de la valeur du paramètre dans la pile
- Appel par référence
 - ☒ Copie de l'adresse de la variable paramètre dans la pile
 - ☒ Tout changement sur la variable est directement visible
- Appel par copie/restauration
 - ☒ Copie de la valeur de la variable dans la pile
 - ☒ Copie dans la variable après exécution de la procédure
 - ☒ Utilisé dans certains langages (*inout* en Ada) (n'existe pas en C)

DESS-ISIDIS

Les RPC

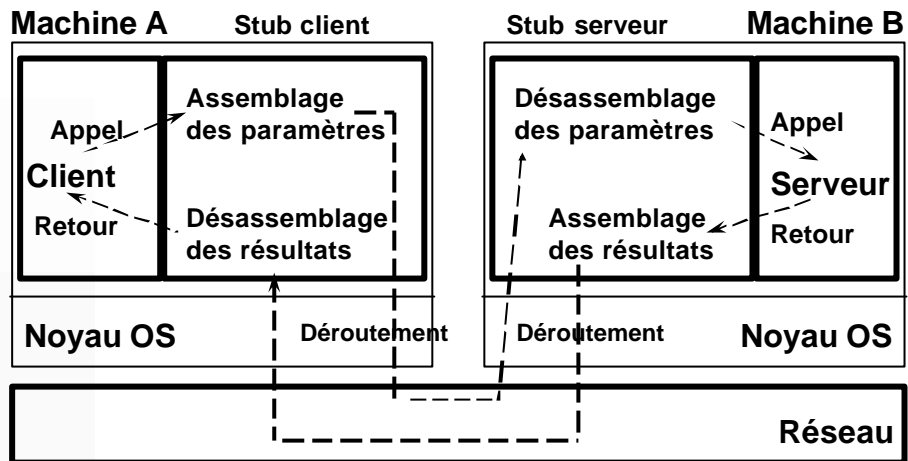
Le modèle RPC

- Même sémantique que le modèle LPC
- Position par rapport à OSI
 - ☒ Couche *session*
- Communication synchrone et transparente
 - ☒ Utilisation transparente de sockets en mode *connecté*
- Différentes implémentations
 - ☒ *DCE-RPC* de l' *Open Software Foundation (OSF)*
 - ☒ *ONC-RPC* de *Sun (NFS, NIS, etc.)*

DESS-ISIDIS

Les RPC

Fonctionnement



DESS-ISIDIS

Les RPC

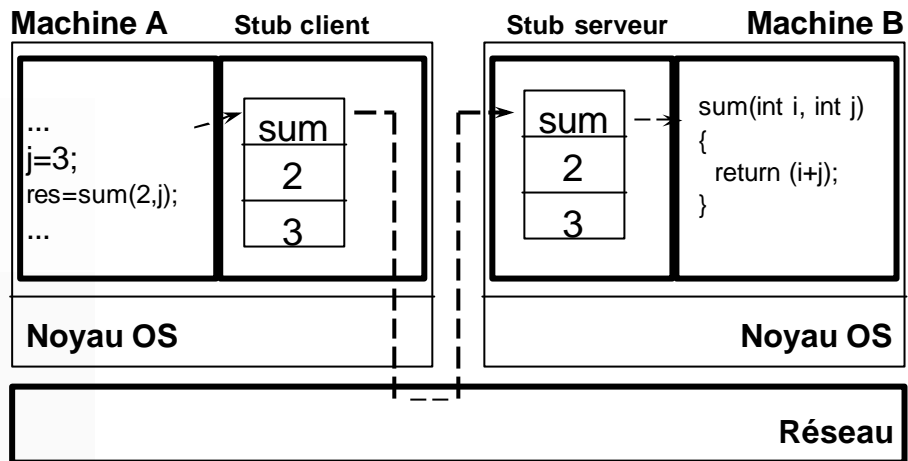
Problèmes

- **Passage de paramètres**
- Identification ou nommage
 - ☒ Localisation (adresse) du serveur
 - ☒ Procédure au sein d'un serveur
- Sémantique des RPC en présence d'échecs

DESS-ISIDIS

Les RPC

Exemple



DESS-ISIDIS

Les RPC

Passage de paramètres

- Passage de paramètres par référence impossible
 - ☒ Passage par valeur
 - ☒ Passage par copie/restauration
- Passage de structures dynamiques (tableaux de taille variable, listes, arbres, graphes, etc.)
- Hétérogénéité des machines
 - ☒ *Byte-ordering* : ordre de stockage des octets différent (*little endian (Intel), big endian (Sun-SPARC)*)
 - ☒ Représentation des arguments : codes des caractères, C1 et C2, virgule flottante, etc.

DESS-ISIDIS

Les RPC

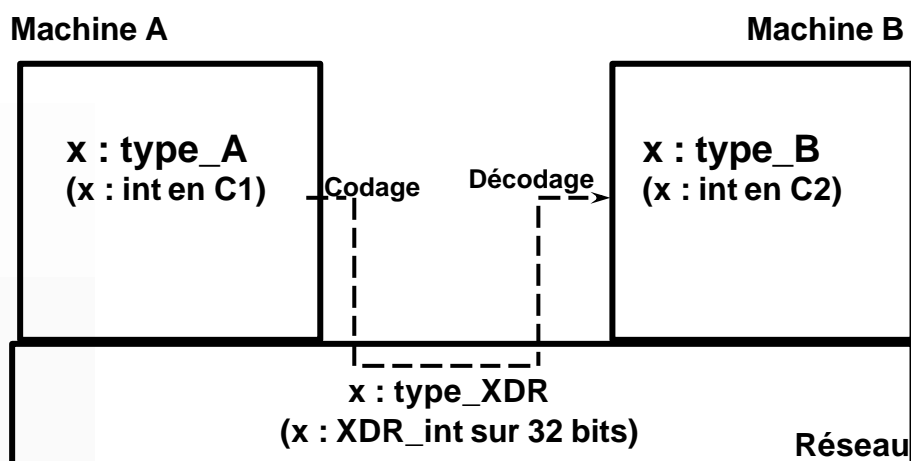
Problème d'hétérogénéité

- Deux solutions possibles
 - ☒ Codage/décodage de *chaque type* de donnée de *toute architecture* à *toute autre architecture*
 - ☒ Format universel intermédiaire (XDR, CDR, etc.)
- Solution de Sun Microsystems
 - ☒ Format *eXternal Data Representation* ou *XDR*
 - ☒ Librairie XDR (types de données XDR + primitives de codage/décodage pour chaque type)
 - ☒ `rpc/xdr.h`

DESS-ISIDIS

Les RPC

XDR : Principe



DESS-ISIDIS

Les RPC

Types de données XDR

type	taille	description
int	32 bits	entier signé de 32 bits
u_int	32 bits	entier non signé de 32 bits
bool_t	32 bits	valeur booléenne (0 ou 1)
enum_t	arb.	type énuméré
hyper	64 bits	entier signé de 64 bits
u_hyper	64 bits	entier non signé de 64 bits
float	32 bits	virgule flot. simple précision
double	64 bits	virgule flot. double précision
opaque	arb.	donnée non convertie
fixed array	arb.	tableau de longueur fixe de n'importe quel autre type
struct_t	arb.	agrégat de données
discriminated union	arb.	structure implémentant des formes alternatives
symbolic constant	arb.	constante symbolique
void	0	utilisé si pas de données
string	arb.	chaîne de car. ASCII

DESS-ISIDIS

Les RPC

XDR : Codage/Décodage (1)

- L'encodage XDR des données contient uniquement les données représentées mais aucune information sur leur type
 - ☒ Si une application utilise un entier de 32 bits le résultat de l'encodage occupera exactement 32 bits et rien n'indiquera qu'il s'agit d'un type entier
 - ☒ Le client et le serveur doivent alors s'entendre sur le format exact des données qu'ils échangent

DESS-ISIDIS

Les RPC

Primitives XDR : Exemple

```
XDR *xdrs;          /* Pointeur vers un buffer XDR */
char buf[BUFSIZE]; /* Buffer pour recevoir les données encodées */
xdr_mem_create (xdrs, buf, BUFSIZE, XDR_ENCODE);
```

```
/* Un buffer stream est créé pour encoder les données
 * chaque appel à une fonction de codage va placer le résultat
 * à la fin du buffer stream ; le pointeur sera mis à jour */
```

```
int i;
...
i=100;
xdr_int(xdrs, &i); /* code l'entier i et le place en fin de buffer stream */
```

Rq : Le programme récepteur décodera les données : xdr_mem_create (... , XDR_DECODE)

DESS-ISIDIS

Les RPC

Autres primitives XDR

Fonction	arguments	type de donnée converti
xdr_bool	xdrs, ptrbool	booléen
xdr_bytes	xdrs, ptrstr, strsize, maxsize	chaîne de caractères
xdr_char	xdrs, ptrchar	caractère
xdr_double	xdrs, ptrdouble	virgule flot., double précision
xdr_enum	xdrs, ptrint	type énuméré
xdr_float	xdrs, ptrfloat	virgule flot. simple précision
xdr_int	xdrs, ip	entier 32 bits
xdr_long	xdrs, ptrlong	entier 64 bits
xdr_opaque	xdrs, ptrchar, count,	données non converties
xdr_pointer	xdrs, ptrobj	pointeur
xdr_short	xdrs, ptrshort	entier 16 bits
xdr_string	xdrs, ptrstr, maxsize	chaîne de caractères
xdr_u_char	xdrs, ptruchar	entier 8 bits non signé
xdr_u_int	xdrs, ptrint	entier 32 bits non signé

opaque : Donnée transmise telle quelle (stockée mais non consultée par un programme donné)

DESS-ISIDIS

Les RPC

Et les structures ???

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

struct sum_request {
    int x; int y;
};
typedef struct sum_request sum_request;

XDR *xdrs; sum_request *objp;
...
if (!xdr_int(xdrs, &objp->x)) return FALSE;
if (!xdr_int(xdrs, &objp->y)) return FALSE;
...
```

DESS-ISIDIS

Les RPC

Problèmes

- Passage de paramètres
- **Identification ou nommage**
 - ☒ Localisation (adresse) du serveur
 - ☒ Procédure au sein d'un serveur
- Sémantique des RPC en présence d'échecs

DESS-ISIDIS

Les RPC

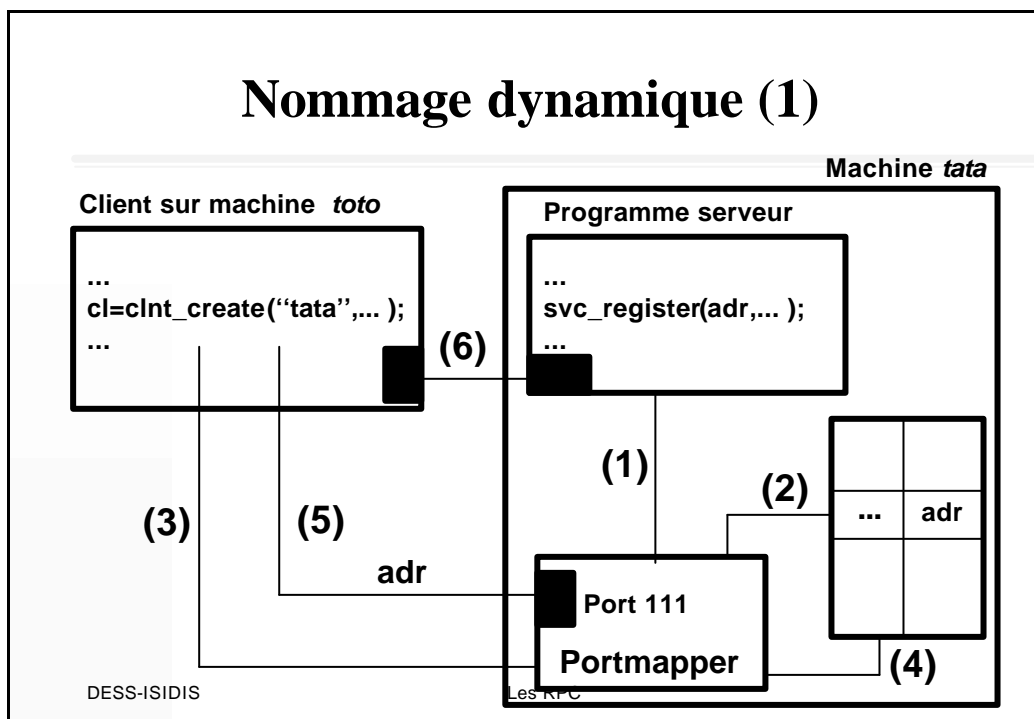
Nommage ou *binding*

- Comment un client fait-il pour trouver le serveur ?
 - ☒ Solution statique : écrit son adresse dans son code
 - ☒ Problème : solution rigide (et si le serveur change d'adresse !!!)
- Solution robuste : nommage dynamique (*dynamic binding*)
 - ☒ Gestionnaire de noms : intermédiaire entre le client et le serveur
 - ☒ **Portmapper** : processus daemon s'exécutant sur le serveur

DESS-ISIDIS

Les RPC

Nommage dynamique (1)



Nommage dynamique (2)

- Le serveur s'enregistre auprès du *portmapper*
 - ☒ Son nom (ou numéro)
 - ☒ Sa version (car il peut en avoir plusieurs)
 - ☒ Son adresse (IP, numéro de port) ou *handle* sur 32 bits
 - ☒ etc.
- Le *portmapper* enregistre ces informations dans sa table de liaisons
- Le client demande l'adresse du serveur au *portmapper* en lui passant le nom et la version du serveur et le protocole de communication à utiliser

DESS-ISIDIS

Les RPC

Adresses de quelques serveurs

Nom	identifieur	description
portmap	100000	port mapper
rstat	100001	rstat, rup, perfmeter
ruserd	100002	remote users
nfs	100003	Network File System
ypserv	100004	Yellow pages (NIS)
mountd	100005	mount, showmount
dbxd	100006	debugger
ypbind	100007	NIS binder
etherstatd	100010	Ethernet sniffer
pcnfs	150001	NFS for PC

0x20000000 - 0x3FFFFFFF : adresses libres
0x00000000 - 0x1FFFFFFF : serveurs officiels

DESS-ISIDIS

Les RPC

Procédure au sein d'un serveur

- Identification

- ☒ Nom (ou numéro) de son programme
- ☒ Version du programme
- ☒ Nom ou numéro de la procédure

DESS-ISIDIS

Les RPC

Problèmes

- Passage de paramètres
- Identification ou nommage
 - ☒ Localisation (adresse) du serveur
 - ☒ Procédure au sein d'un serveur
- **Sémantique des RPC en présence d'échecs**

DESS-ISIDIS

Les RPC

Différentes classes d'échecs (1)

- Le client est incapable de localiser le serveur
 - ☒ Le serveur est en panne
 - ☒ L'interface du serveur a changé
 - ☒ Solutions : retourner -1 !!!, exceptions, signaux
- La requête du client est perdue
 - ☒ Temporisation et ré-émission de la requête
- La réponse du serveur est perdue
 - ☒ Temporisation et ré-émission de la requête par le client

DESS-ISIDIS

Les RPC

Différentes classes d'échecs (2)

- ☒ Problème : risque de ré-exécuter la requête plusieurs fois (opération bancaire !!!!)
- ☒ Solution : un bit dans l'en-tête du message indiquant s'il s'agit d'une transmission ou retransmission
- Le serveur tombe en panne après réception d'une requête
 - ☒ (i) Après exécution de la requête et envoi de réponse
 - ☒ (ii) Après exécution de la requête, avant l'envoi de la réponse
 - ☒ (iii) Pendant l'exécution de la requête
 - ☒ Comment le client fait-il la différence entre (ii) et (iii) ?

DESS-ISIDIS

Les RPC

Différentes classes d'échecs (4)

☒ Trois écoles de pensée (sémantiques)

- ☞ Sémantique *une fois au moins* : le client ré-émet jusqu'à avoir une réponse (RPC exécuté au moins une fois)
- ☞ Sémantique *une fois au plus* : le client abandonne et renvoie un message d'erreur (RPC exécuté au plus une fois)
- ☞ Sémantique *ne rien garantir* : le client n'a aucune aide (RPC exécuté de 0 à plusieurs fois)

■ Le client tombe en panne après envoi d'une requête

- ☒ Requête appelée *orphelin*. Que doit-on en faire ?

DESS-ISIDIS

Les RPC

Différentes classes d'échecs (5)

☒ Solutions de *Nelson*

- ☞ Extermination : Le client utilise un journal de trace et tue les orphelins : solution coûteuse en espace et complexe (orphelins d'orphelins !!!)
- ☞ Réincarnation : Définition de périodes d'activité incrémentale du client. Après une panne, il diffuse un message indiquant une nouvelle période. Ses orphelins sont détruits.
- ☞ Réincarnation douce : variante de la précédente. Un orphelin est détruit seulement si son propriétaire est introuvable.
- ☞ Expiration : Chaque RPC dispose d'un quantum q de temps pour s'exécuter. Il est détruit au bout de ce quantum. Pb : valeur de q ?
- ☞ Problème de ces solutions : si l'orphelin détruit a verrouillé des ressources ?!!!

DESS-ISIDIS

Les RPC

Méthodologie de développement à base de RPC

DESS-ISIDIS

Les RPC

Composants d'une application RPC

■ Client

- ☒ Localiser le serveur et s'y connecter
- ☒ Faire des appels RPC (requêtes de service)
 - ☒ Emballage des paramètres
 - ☒ Soumission de la requête
 - ☒ Déemballage des résultats

Stub client
+
Primitives XDR

■ Serveur

- ☒ S'enregistrer auprès du *portmapper*
- ☒ Attendre les requêtes du client et les traiter
 - ☒ Déemballage des paramètres
 - ☒ Appel local du service (procédure) demandé(e)
 - ☒ Emballage des résultats

Stub serveur
+
Primitives XDR

DESS-ISIDIS

Les RPC

Besoins

- Le client a besoin de connaître le nom symbolique du serveur (numprog,numver) et ses procédures
 - ☒ Publication dans un **contrat** (fichier .x)
 - ☒ Langage *RPCL*
- Le serveur a besoin des implémentations des procédures pour pouvoir les appeler
 - ☒ Fichier contenant les implémentations des procédures
- Serveur et client ont besoin des stubs de communication
 - ☒ Communication transparente ==> génération automatique des stubs et des fonctions XDR de conversion de paramètres
 - ☒ *RPCGEN* (compilateur de contrat) + *Runtime RPC*

DESS-ISIDIS

Les RPC

Le langage RPCL

- Constantes
 - ☒ *Const* nom=valeur;
- Enumérations et structures
 - ☒ Idem C
- Unions
 - ☒ Idem structures avec variantes en Pascal
- Tableaux
 - ☒ <type_de_base> <nom_tab> <TAILLE> (ex : int toto<MAXSIZE >)
- Définition de types
 - ☒ typedef

DESS-ISIDIS

Les RPC

Forme générale du contrat

```
/* Définitions de types utilisateur */
...
program «nomprog» {
  version «nomversion1» {
    «typeres1» PROC1(«param1») = 1;
    ...
    «typeresn» PROCn(«paramn») = n;
  } = 1;
  ...
  version «nomversionm» {
    ...
  } = m;
} = «numéro_du_programme»;
```

DESS-ISIDIS

Les RPC

Méthodologie de développement

- Ecrire le contrat *toto.x* dans le langage RPCL
- Compiler le contrat avec RPCGEN
 - ☒ *toto.h* : déclarations des constantes et types utilisés dans le code généré pour le client et le serveur
 - ☒ *toto_xdr.c* : procédures XDR utilisés par le client et le serveur pour encoder/décoder les arguments,
 - ☒ *toto_clnt.c* : procédure *stub* côté client
 - ☒ *toto_svc.c* : procédure *stub* côté serveur
- Ecrire le client (*client.c*) et le serveur (*serveur.c*)
 - ☒ *Serveur.c* : implémentation de l'ensemble des procédures

DESS-ISIDIS

Les RPC

Compilation et exécution

■ Compilation

- ☒ (g)cc serveur.c toto_svc.c toto_xdr.c -o Serveur
- ☒ (g)cc client.c toto_clnt.c toto_xdr.c -o Client

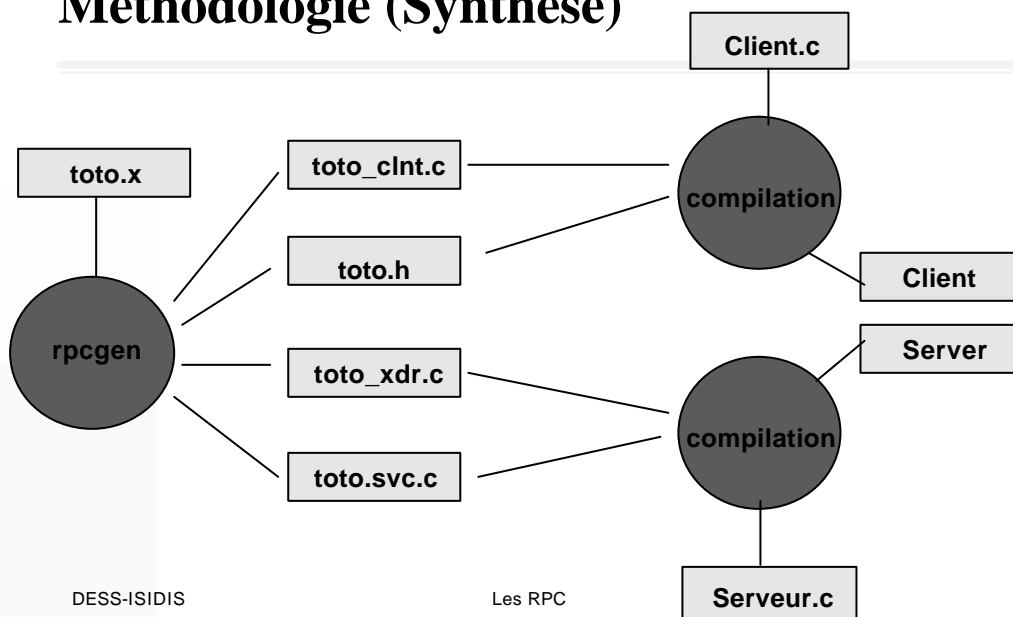
■ Exécution

- ☒ rsh «host» Serveur
- ☒ Client «host» «liste d'arguments»

DESS-ISIDIS

Les RPC

Méthodologie (Synthèse)

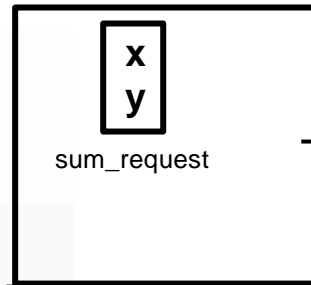


DESS-ISIDIS

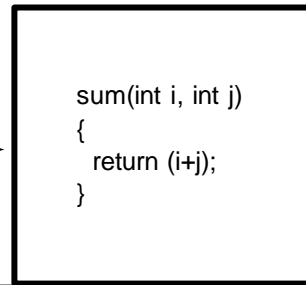
Les RPC

Exemple : sum

Client sur Machine A



Serveur sur Machine B



DESS-ISIDIS

Les RPC

Le contrat (sum.x)

```
struct sum_request {
    int x; int y;
};

program SUMPROG {
    version SUMVERS {
        int SUM(sum_request) = 1;
    } = 1; /* numéro de version du programme */
} = 0x2000009a; /* numéro du programme */
```

DESS-ISIDIS

Les RPC

Fichiers générés avec RPCGEN

- sum.h, sum_xdr.c, les stubs sum_clnt.c et sum_svc.c

- sum.h

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

struct sum_request {
    int x; int y;
};
typedef struct sum_request sum_request;
bool_t xdr_sum_request();

#define SUMPROG ((u_long)0x2000009a)
#define SUMVERS ((u_long)1)
#define SUM ((u_long)1)
extern int *sum_1();
```

DESS-ISIDIS

Les RPC

sum_xdr.c

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

struct sum_request {
    int x; int y;
};
typedef struct sum_request sum_request;

bool_t xdr_sumrequest (XDR *xdrs, sum_request *objp)
{
    if (!xdr_int(xdrs, &objp->x)) return FALSE;
    if (!xdr_int(xdrs, &objp->y)) return FALSE;
    return TRUE;
}
```

DESS-ISIDIS }

Les RPC

sum_clnt.c

```
...
#include <sys/socket.h>
#include <rpc/rpc.h>
#include "sum.h"

static struct timeval TIMEOUT = {25, 0};
int* sum_1 (sum_request *argp, CLIENT *clnt)
{
    static int res;
    bzero((char *)&res, sizeof(res));
    if (clnt_call(clnt, SUM, (xdrproc_t)xdr_sum_request, argp,
        (xdrproc_t)xdr_int, &res, TIMEOUT ) != RPC_SUCCESS)
        return NULL;
    return (&res);
}
DESS-ISIDIS Les RPC
```

sum_svc.c

```
...
#include <sys/socket.h>
#include <rpc/rpc.h>
#include "sum.h"

static void sumprog_1();
main()
{
    SVCXPRT *transp;
    transp = svctcp_create(RPC_ANYSOCK, 0, 0);
    if (svc_register(transp, SUMPROG, SUMVERS, sumprog_1,
        IPPROTO_TCP)) exit(1);
    svc-run();
}
DESS-ISIDIS Les RPC
```

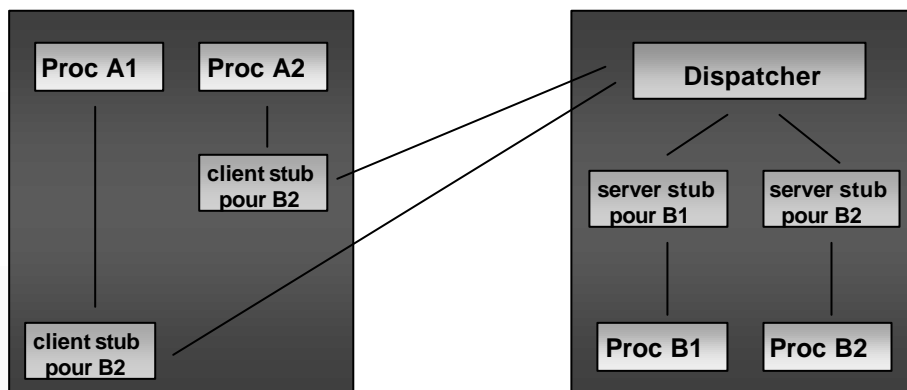
sum_svc.c (suite)

```
static void sumprog_1(struct svc_req *rqstp, SVCXPRT *transp)
{
    /* Dispatcher */
    ...
    switch (rqstp->rq_proc) {
        ...
        case SUM:
            /* Construire la requête */
            /* Appeler la fonction sum_1(...) */
            /* Construire la réponse */
            /* Envoyer la réponse au client : svc_sendreply */
            /* Libérer l'espace utilisé dynamiquement */
        }
    }
}
```

DESS-ISIDIS

Les RPC

Le dispatcher en général



DESS-ISIDIS

Les RPC

Serveur.c

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

Un seul paramètre

```
int *sum_1(sum_request *req)
{
    static int s;

    s=req->x + req->y;
    return &s;
}
```

DESS-ISIDIS

Les RPC

Client.c

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

Procédure d'interface

```
CLIENT *cl;
int sum(int x, int y)
{
    int *s;
    sum_request req;
    req.x=x; req.y=y;
    s=sum_1(&req,cl);
    return *s;
}
```

Procédure de communication

DESS-ISIDIS

Les RPC

Client.c (suite)

```
main (int argc, char **argv)
{
    int s;
    sum_request req;

    if (argc != 4 ) { printf("USAGE: %s host x y \n", argv[0]); exit(1);}
    cl = clnt_create(argv[1], SUMPROG, SUMVERS, "tcp");
    if (cl == NULL) { printf("Le serveur ne répond pas \n"); exit(1); }
    s=sum(atoi(argv[2]), atoi(argv[3]));
    printf("La somme est %d \n",s);
}
```