Appel de procédures distantes dans le modèle client-serveur

Jonathan Lejeune

Sorbonne Université/LIP6-INRIA

SRCS – Master 1 SAR 2017/2018

sources :

UNIX : programmation et communication, J-M. Rifflet , J-B Yunès

cours de Bertil Foliot

Exemple fil rouge

Objectif

Développer un service pour calculer des divisions.

Spécifications de fonctionnalités

- Division classique entre deux flottants renvoyant un flottant
- Division entière entre deux entiers renvoyant deux valeurs : quotient et reste

Ce que nous offre le réseau

Protocole de transport

Couche réseau permettant :

- la communication de bout en bout entre deux processus distant
- le multiplexage des communications entre deux machines

Identification d'un processus sur le réseau

- Identification de la machine : adresse IP ou nom DNS
- Un identifiant locale à la machine : numéro de port

Les protocoles de transport

User Datagram Protocol

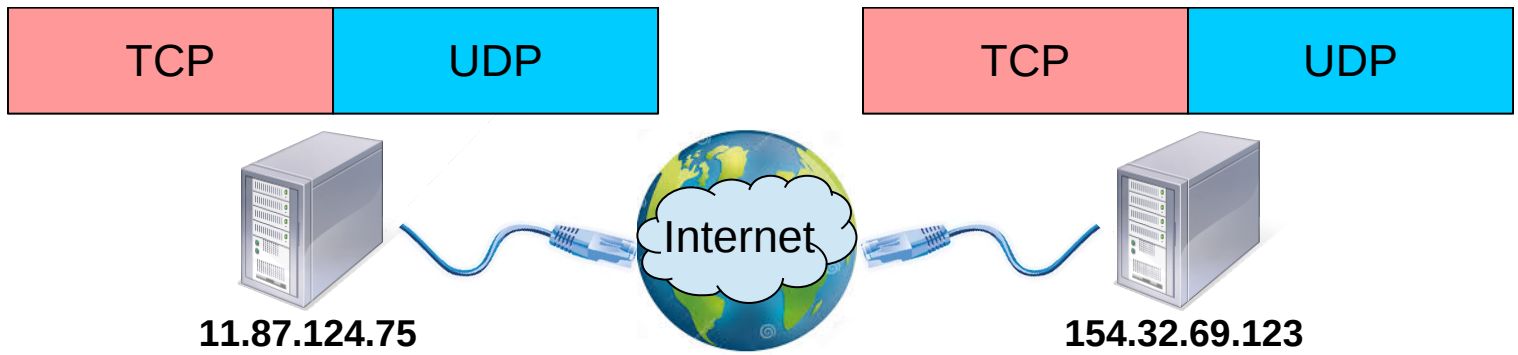
- Non connecté
- Non fiable

Services UDP connus : echo (1), daytime (13), DNS (53), sunrpc (111)

Transport Control Protocol

- Connecté
- Communication fiable et FIFO

Services TCP connus : ftp (20,21), ssh (22), http (80), sunrpc (111)



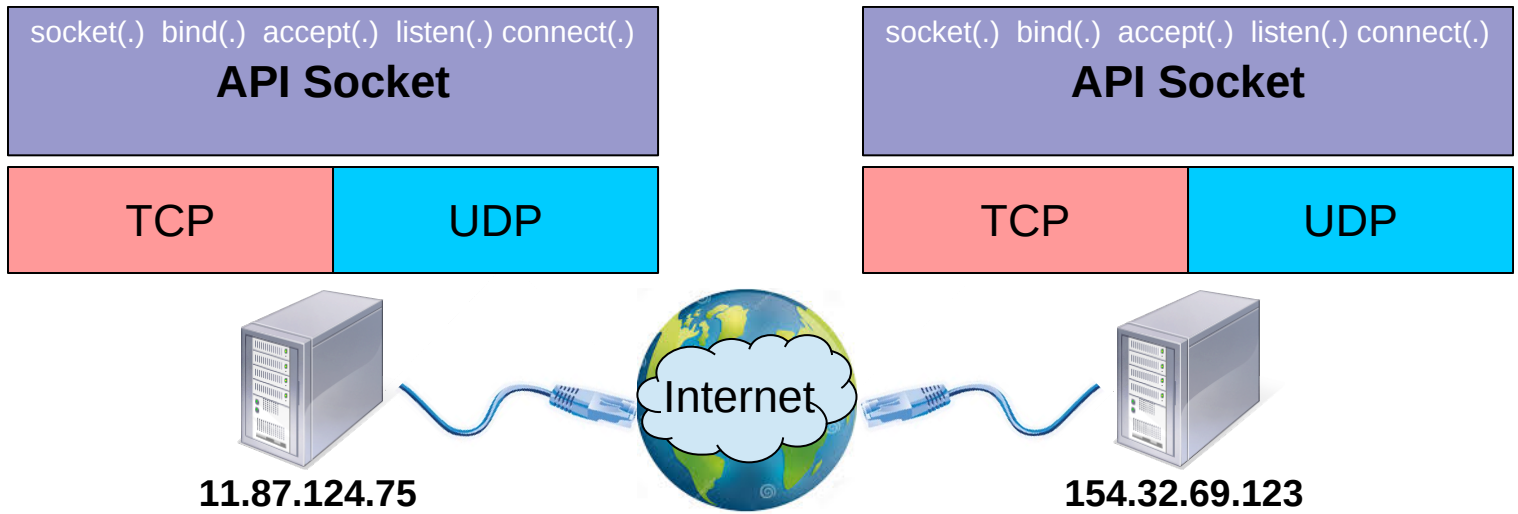
Les sockets

Définition

API système permettant la communication bidirectionnelle entre deux processus pouvant être distant.

Caractéristiques

- Type : datagramme (`SOCK_DGRAM`) ou flux connecté (`SOCK_STREAM`)
- Domaine : local (`AF_UNIX`) ou Internet (`AF_INET`)



Notre fil rouge avec les sockets

Ma TODO list

- Décider d'un protocole applicatif (messages, sérialisation des données)
- Coder la connexion/déconnexion un client et un serveur
- Coder le code métier du client et du serveur
- Gérer les erreurs, les fautes

Fil rouge : Messages du protocole applicatif

```
enum typemess{DIV,DIV_ENTIERE, REP_DIV, REP_DIV_ENTIERE}
```

```
typedef struct {
  enum typemess type;
  double a;
  double b;
} Div;
```

```
typedef struct {
   enum typemess type;
   int a;
   int b;
} DivEnt;
```

```
typedef struct {
   enum typemess type;
   double res;
} RepDiv ;
```

```
typedef struct {
   enum typemess type;
   int q;
   int r;
} RepDivEnt;
```

Fil rouge : Code client/serveur (socket TCP)

Coté client

```
int main(int argc, char** argv){
 gethostbyname(..)//nom serveur
 socket(..)//creer socket
 bind(..)//liaison socket-port
 connect(..)//joindre  serveur
 switch(argv[2]){
   case DIV:
     //send Div
     //recv RepDiv
     break;
   case DIV_ENTIERE:
     //send DivEnt
     //recv RepDivEnt
     break;
 }
 close(..)//fermeture socket
}
```

Coté serveur

```
int main(int argc, char** argv){
 socket(..)//creer socket
 bind(..)//liaison socket-port
 listen(..)//socket d'écoute
 while(1){
  accept(..)//attente de cnx
  //recv sur socket connexion
  switch(type){
   case DIV:
    RepDiv r;
    r.res=div.a / div.b;
    //send rep_div with r
    break;
   case DIV_ENTIERE:
    RepDivEnt r;
    r.q = div.a/ div.b;
    r.r = div.a % div.b
    //send rep_div_entiere r
    break;
   }
   close(..);//close connexion
  }
 }
```

Inconvénients de l'API des Socket

Une programmation trop lourde

- Peu intuitif
- Programmation dépendante des primitives réseaux (rcv, send)
- Gestion manuelle des fautes, des acquittements de messages
- Définition manuelle d'un protocole applicatif
- Pas de gestion de l'hétérogénéité $\Rightarrow$ code dépendant :
  - de la machine utilisée (architecture, endianness, nombre flottant)
  - du langage de programmation
  - de l'OS (Unix, Windows, ...)

Besoin

$\Rightarrow$ **d'un modèle plus simple**

$\Rightarrow$ **d'une standardisation de représentation des données**

Point de départ : Invocation de fonctionnalité locale

Fonctionnement d'un appel de procédure/fonction

- Transfert des données : copie des arguments sur la pile
- Transfert de contrôle : jump sur la fonction appelée

Un modèle simple

Appelant et appelé :

- même processus : en cas de faute appelant et appelé s'arrêtent
- même système : Homogénéité du langage, de la représentation des données
- même espace d'adressage : pas besoin d'I/O

Vers un modèle d'invocation distante

Objectif

Permettre à une entité active d'un système d'invoquer un service en réalisant un appel de fonction dont l'exécution aura lieu dans une autre entité d'un système distant.

Problématiques

- Comment gérer les fautes ?
  - Que faire en cas de non-réponse ?
  - Quelle sémantique en cas ré-réémission ?
- Comment gérer l'hétérogénéité ?
- Comment identifier les différentes versions du serveurs/ du client ?
- Comment identifier les procédures sur le serveur ?

$\Rightarrow$ **Protocole RPC**

Remote Procedure Call (RPC)

Qu'est ce que c'est

Un protocole de communication inter-processus de haut-niveau se basant sur les sockets

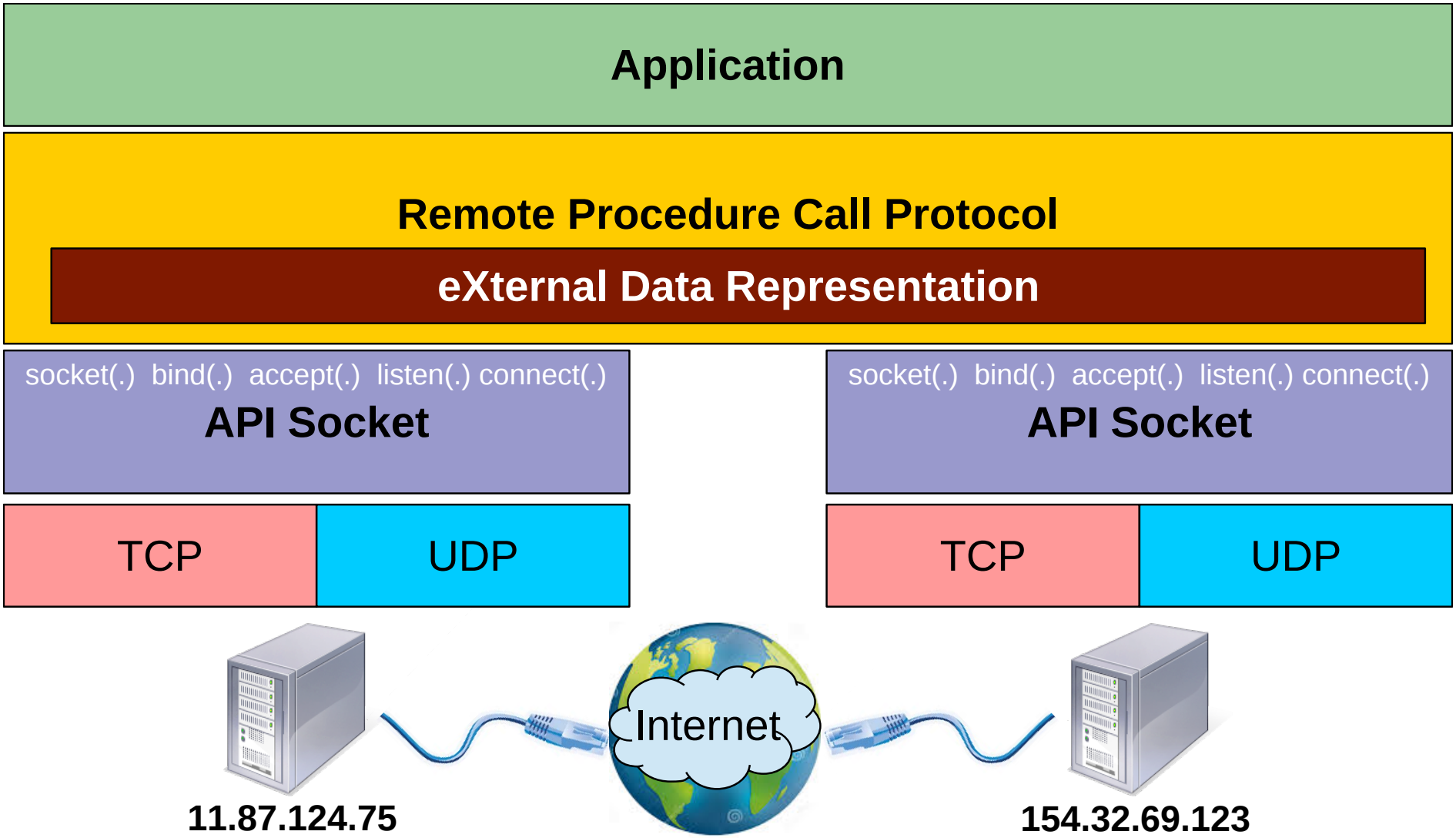
Avantages

- Communication simple
- Transparence des couches réseaux
- Gestion de l'hétérogénéité (standard XDR)

Inconvénient

- Sémantique complexe en cas de panne

Remote Procedure Call (RPC)



Fonctionnement général d'une invocation distante

Coté client

- Encodage des paramètres
- Localiser le serveur
- Envoie d'un message
- Attendre la réponse
- Décoder le résultat

Coté serveur

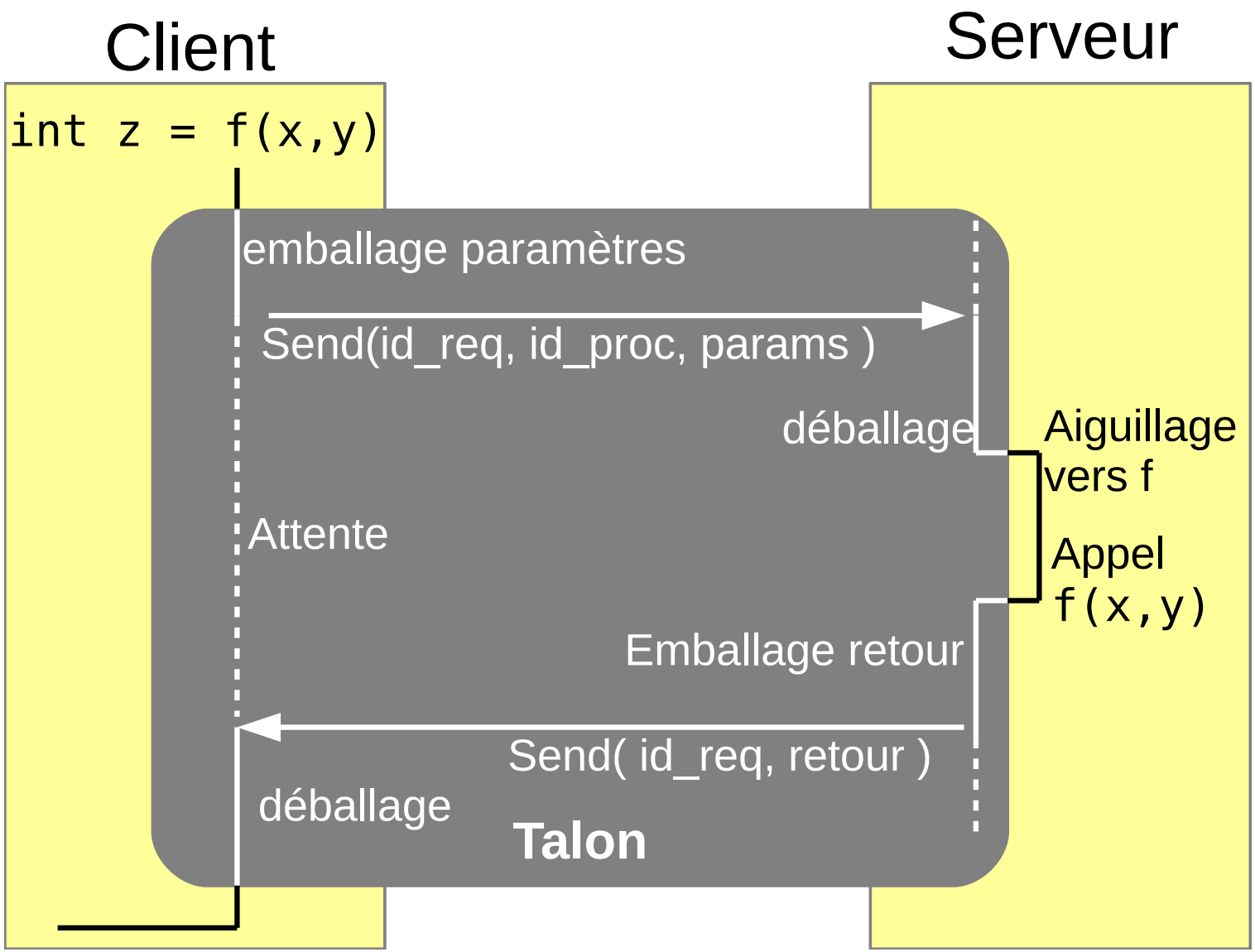
- Identification de la procédure
- Décodage des paramètres
- Exécution
- Encodage du retour
- Envoi de la réponse

Le talon

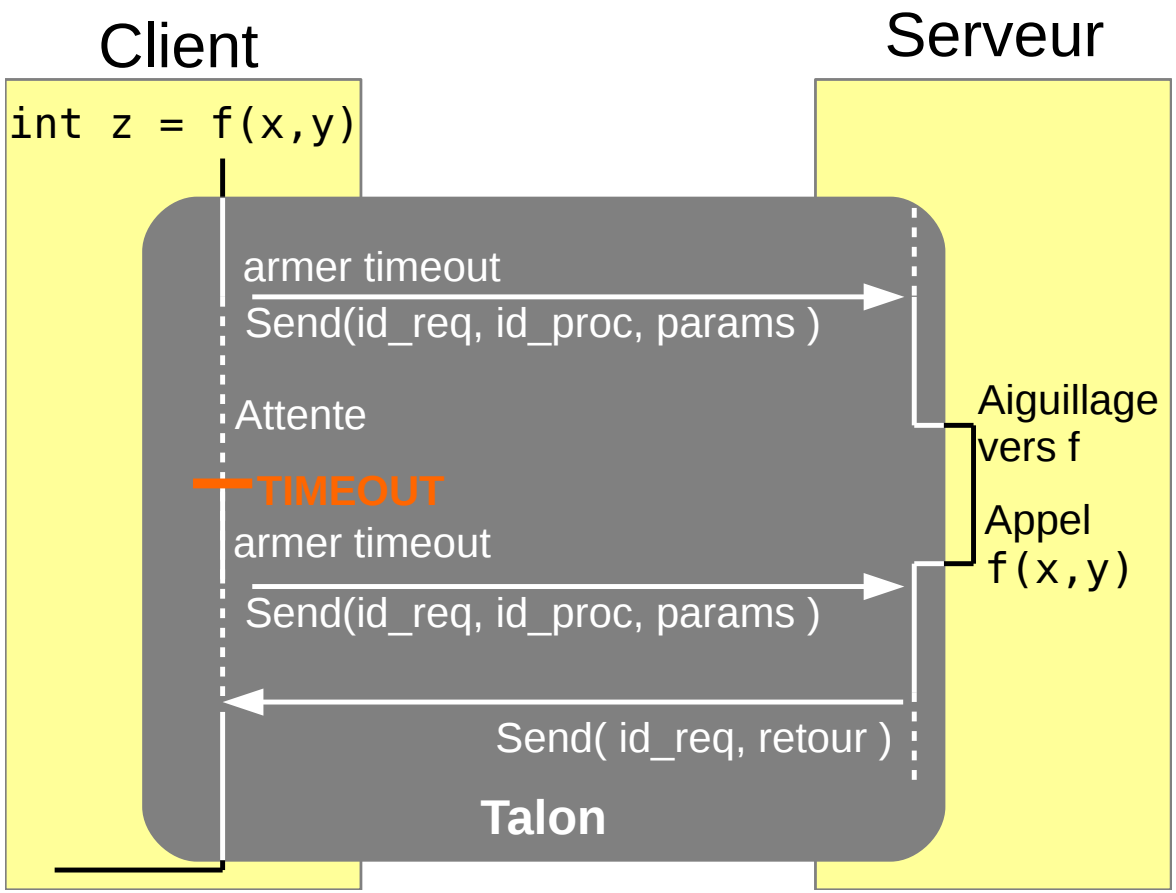
Brique logicielle permettant :

- de représenter le serveur pour le client
- de représenter le client pour le serveur
- l'emballage des paramètres à l'envoi
- le déballage des paramètres à la réception

Schéma Fonctionnement général d'une invocation distante



Traitement des défaillances (UDP)



Mécanisme de délai de garde

- Armement d'un timeout à l'envoi d'un message
- Réémission du message au déclenchement du Timeout

Sémantique du traitement des défaillances

Combien de fois le traitement a été effectué ?

- aucune ?
- exactement une fois (idéal) ?
- au moins une fois ?

  cas des RPC-SUN : réémission jusqu'à réception d'une réponse

Attention à la valeur du timeout

- Trop faible valeur $\Rightarrow$ plusieurs exécutions de la même requête
- Trop grande valeur $\Rightarrow$ latence de détection

Normalisation des données

Le protocole XDR

- Permettre entre deux processus l'échange de données :
  - de type quelconque
  - en passant par une représentation standard

Contraintes imposées à une machine implémentant XDR

Pouvoir convertir n'importe quel objet typé :

- de sa représentation locale vers sa représentation XDR
- de sa représentation XDR vers sa représentation locale

Inconvénient

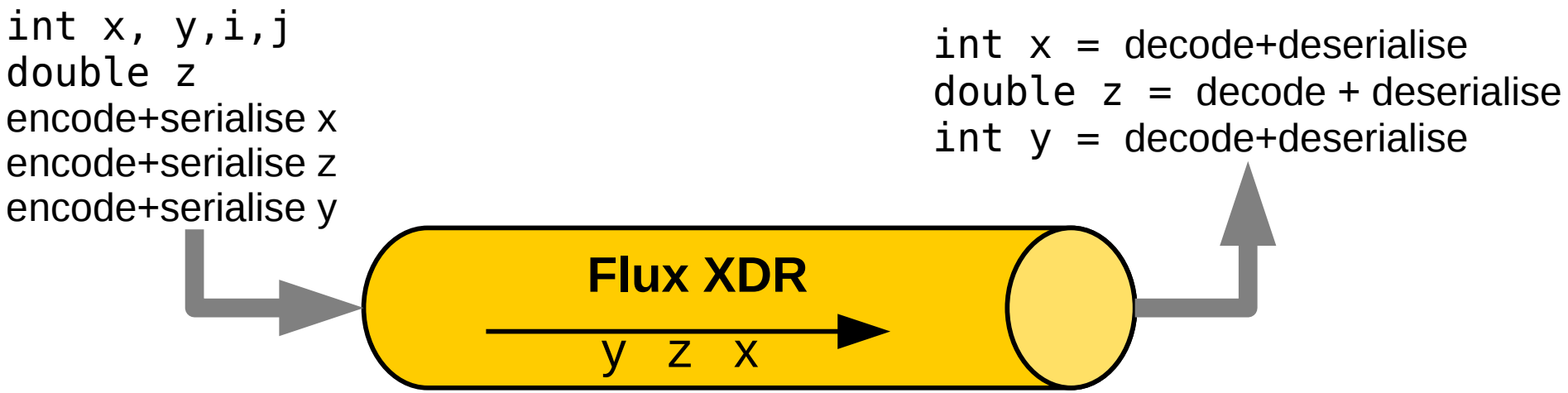
Conversion inutile si deux machines ont la même représentation locale

Principes du protocole XDR

Opérations de base

Produire un flux continu d'octets encodé par :

- la Sérialisation et l'encodage XDR des données coté expéditeur
- la Désérialisation et le décodage XDR des données coté destinataire



Construction/destruction de flux XDR

Déclaration

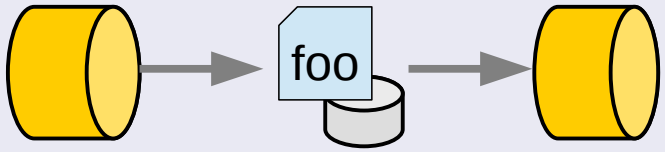
```
XDR xdr; //declaration d'un descripteur de flux XDR
```

Construction à partir des IO standards

```
void xdrstdio_create(XDR *xdrs, FILE *file, enum xdr_op op);
```

```
FILE *f = fopen("foo",0666);
xdrstdio_create(&xdr,f,XDR_ENCODE);
```

```
FILE *f = fopen("foo",0666);
xdrstdio_create(&xdr,f,XDR_DECODE);
```

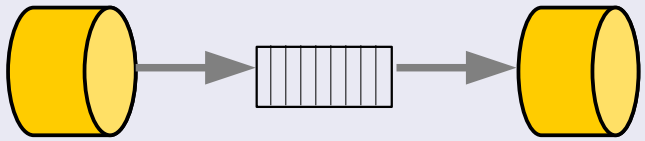


Construction sur un tampon mémoire

```
void xdrmem_create(XDR *xdrs, char* adr, u_int size, enum xdr_op op);
```

```
char buf[256];
xdrmem_create(&xdr,buf,256,XDR_ENCODE);
```

```
xdrmem_create(&xdr,buf,256,XDR_DECODE);
```



Destruction

```
xdr_destroy(&xdr); //liberation de la memoire
```

Conversions de base

Le type xdrproc_t

Pointeur de fonction sur un filtre XDR

```
typedef bool_t (*xdrproc_t)(XDR*, type *)
```

Type C	Filtre XDR correspondant
`char`	`bool_t xdr_char(XDR *, char *);`
`short`	`bool_t xdr_short(XDR *, short *);`
`int`	`bool_t xdr_int(XDR *, int *);`
`long`	`bool_t xdr_long(XDR *, long *);`
`unsigned short`	`bool_t xdr_ushort(XDR *, unsigned short *);`
`unsigned int`	`bool_t xdr_uint(XDR *, unsigned int *);`
`unsigned long`	`bool_t xdr_ulong(XDR *, unsigned long *);`
`float`	`bool_t xdr_float(XDR *, float *);`
`bool_t`	`bool_t xdr_enum(XDR *, bool_t *);`
`enum enum_t`	`bool_t xdr_enum(XDR *, enum enum_t *);`

Conversions plus élaborées

- **Conversion de chaine de caractère :**

```
bool_t xdr_string(XDR *xdr, char **s, u_int length);
```

- **Conversion de tableau d'octets :**

```
bool_t xdr_bytes(XDR *xdr, char **tab, u_int size, u_int maxsize);
```

- **Conversion de tableau de taille fixe d'objets :**

```
bool_t xdr_vector(XDR *xdr, char *tab, u_int nb_elem,
                  u_int size_elem, xdrproc_t  fct);
```

- **Conversion de tableau de taille inconnu d'objets :**

```
bool_t xdr_array(XDR *xdr, char **tab, u_int* nb_elem,
        u_int max_nb_elem, u_int size_elem, xdrproc_t fct
            );
```

- **Conversion d'un objet référencé par un pointeur :**

```
bool_t xdr_reference(XDR *xdr, void **ptr
                     , u_int size_elem, xdrproc_t  fct);
```

Conversion de structures

Principe

Définir une fonction de type `xdrproc_t` qui appelle séquentiellement les fonctions de conversion de chacun de ses membres

```
typedef struct{
        type1 champ1;
        type2 champ2;
        ...
        typeN champN;
}monType;
...
bool_t xdr_monType(XDR* xdr, monType * objet){

        return     xdr_type1(xdr, &objet->champ1)
                && xdr_type2(xdr, &objet->champ2)
                && ...
                && xdr_typeN(xdr, &objet->champN);

}
```

Fil rouge : standardisation des données

Modification des structures

Les filtres XDR permettent de typer les données du flux :

- Suppression des `enum typemess` et création d'un filtre XDR par type de message

```
typedef struct {
    enum typemess type;
    double a;
    double b;
} Div;
```

```
typedef struct {
    enum typemess type;
    int a;
    int b;
} DivEnt;
```

```
typedef struct {
    enum typemess type;
    double res;
} RepDiv;
```

```
typedef struct {
    enum typemess type;
    int q;
    int r;
} RepDivEnt;
```

Fil rouge : standardisation des données

Ajout de fonctions

Création des filtres XDR.

```
bool_t xdr_Div (XDR *xdr, Div *objp){
return xdr_double(xdr, &objp->a)
      && xdr_double(xdr, &objp->b);
}

bool_t xdr_DivEnt (XDR *xdr, DivEnt *objp){
        return xdr_int(xdr, &objp->a)
      && xdr_int(xdr, &objp->b);
}

bool_t xdr_RepDivEnt (XDR *xdr, RepDivEnt *obj){
    return xdr_int(xdr, &objp->q)
      && xdr_int(xdr, &objp->r);
}
```

L'implémentation Sun RPC

Présentation

- Implémentation des RPC par Sun années 80 (mais toujours d'actualité)
- Application phare : le NFS
- Se base sur les sockets et XDR

Outils logiciels associés

- Un mécanisme d'identification de procédure
- Un service de nommage : `portmap`
- Deux niveaux de développement :
  - Couche haute : construction de clients/serveurs simples
  - Couche basse : construction de clients/serveurs plus élaborés
- RPCL : un langage de spécifications et son compilateur (rpcgen)

Identification des procédures

Un quadruplet d'entier de 32 bits

- L'adresse IP de la machine serveur
- Un numéro de programme au sein de la machine
- Un numéro de version du programme
- Un numéro de procédure au sein du programme

Comment choisir ces valeurs ?

- @IP → en fonction de la machine serveur choisie
- num prog → Affectés en fonction d'une plage de valeurs :
  - de `00.00.00.00` à `1f.ff.ff.ff` : Sun
  - de `20.00.00.00` à `3f.ff.ff.ff` : Admin local
  - de `40.00.00.00` à `5f.ff.ff.ff` : Développeur
  - de `60.00.00.00` à `ff.ff.ff.ff` : réservé
- num version et procédure → par le programmeur du serveur

Le fichier de services /etc/rpc

Role

Base de données des services standard sur le système où elle est consultée

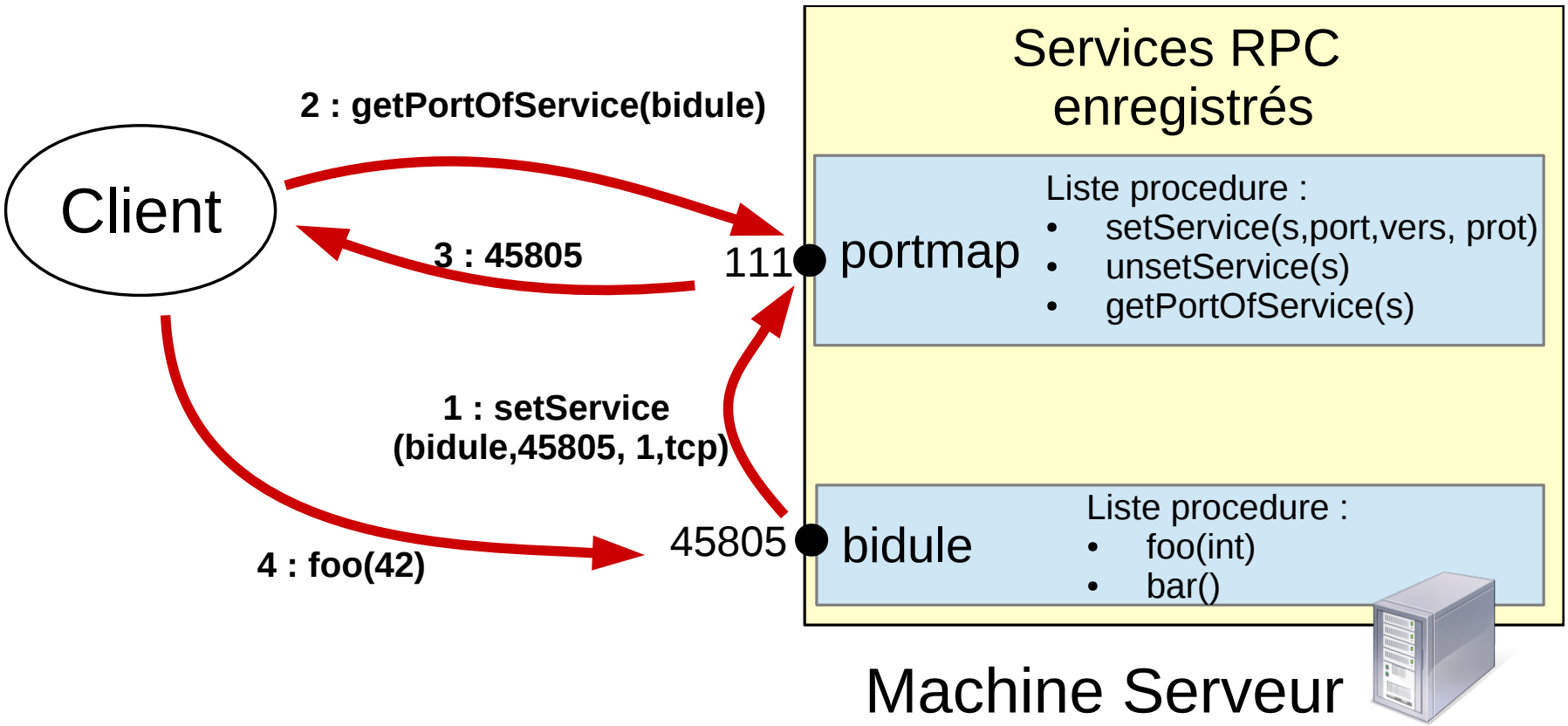
```
$ cat /etc/rpc
#nameprog  id_prog  alias_names

portmapper 100000   portmap sunrpc
rstatd     100001   rstat rstat_svc rup perfmeter
rusersd    100002   rusers
nfs        100003   nfsprog
ypserv     100004   ypprog
mountd     100005   mount showmount
ypbind     100007
walld      100008   rwall shutdown
yppasswdd  100009   yppasswd
....
```

Le service portmap (ou rpcbind ou sunrpc)

Caractéristiques

- Service RPC permettant d'associer dynamiquement un numéro de programme avec un port d'écoute et un protocole de transport.
- Écoute sur le port 111 sur UDP et TCP



Interagir avec le service portmap

La commande rpcinfo

- `rpcinfo -p <host>` : liste des services RPC enregistrés sur `host`
- `rpcinfo -t <host> <port>` : test si il existe un service RPC à l'écoute du port `port` en tcp sur la machine `host` (-u pour udp)
- `rpcinfo -d <num_prog> <num_version>` : désactiver un service RPC

```
lejeune@ssh:~$ rpcinfo -p localhost
   program no_version protocole  no_port
    100000    2          tcp        111  portmapper
    100000    2          udp        111  portmapper
    100024    1          udp      46661  status
    100024    1          tcp      49043  status
    100021    1          udp      46178  nlockmgr
    100021    3          udp      46178  nlockmgr
    100021    1          tcp      38203  nlockmgr
    100021    3          tcp      38203  nlockmgr
```

Couche haute : point de vue serveur

Conditions pour qu'une fonction devienne un service

- Les paramètres de la fonction doivent être rassemblés dans une structure
- si la fonction renvoie un résultat, faire en sorte que le résultat soit renvoyé sous la forme d'un pointeur en zone statique

Enregistrer un service : la fonction `registerrpc`

```
bool_t registerrpc(u_long programme,//id programme
                   u_long version,// id version du programme
                   u_long id_proc,// id de la fonction
                   char* (*f)(void),// pointeur vers la fonction
                   xdrproc_t xdrdata, //filtre xdr des arguments
                   xdrproc_t xdrresult ) // filtre xdr du
                       resultat.
```

Attendre les requêtes sur une socket de connexion

```
void svc_run();
```

Fil rouge : les services en couche haute

```
static double res;

void* div(Div* div){
 res = div.a / div.b;
 return (void *) &res;
}
```

```
static RepDivEnt rep_de;

void* divent(DivEnt* de){
 rep_de.q=de.a/de.b;
 rep_de.r=de.a%de.b;
 return (void*) &rep_de;
}
```

Fil rouge : code fonction main du serveur (couche haute)

```
#define DIV_PROG 0x40000003
#define DIV_V1 1
#define DIV 1
#define DIVENT 2
```

```
int main(){
 pmap_unset(DIV_PROG,DIV_V1);//desenregistre par securite

 registerrpc(DIV_PROG, DIV_V1, DIV,
             div, xdr_Div, xdr_RepDiv);

 registerrpc(DIV_PROG, DIV_V1, DIVENT,
             divent, xdr_DivEnt, xdr_RepDivEnt);
 svc_run();
}
```

Couche haute : point de vue client

Appeler une procédure distante : la fonction `callrpc`

```
int callrpc(char* machine          // nom de la machine serveur
            u_long programme,      // id programme
            u_long version,        // id version du prog
            u_long id_proc,        // id de procedure
            xdrproc_t xdrdata,     // filtre xdr des arguments
            char* data,            // pointeur vers les arguments
            xdrproc_t xdrresult,   // filtre xdr du résultat
            char* resultat );      //pointeur vers le résultat
```

Appel bloquant jusqu'à réception de la réponse

Fil rouge : fonctions coté client

```
double div(char* machine, double a, double b){
 double rep;
 Div d;
 d.a=a; d.b=b
 callrpc(machine, DIV_PROG, DIV_V1, DIV, xdr_div, &d, xdr_double, &
     rep);
 return rep;
}
```

```
int divent(char* machine, int a, int b, int* reste){
 DivEnt de;
 de.a=a; de.b=b;
 RepDivEnt res;
 callrpc(machine, DIV_PROG, DIV_V1, DIV, xdr_DivEnt, &de,
     xdr_RepDivEnt, &res);
 *reste=res.r;
 return res.q;
}
```

Fil rouge : fonction main du client

```
int main(){
 double x= div("serveur.adresse.fr", 10.56, 5.23);
 int reste;
 int y =divent("serveur.adresse.fr", 99, 56, &reste);

 printf("␣x␣=␣%f␣y=%d␣␣reste=%d", x,y,reste);
}
```

Couche basse : Point de vue serveur

Le type SVCXPRT

Descripteur de services encapsulant :

- descripteur de la socket d'écoute
- le numéro de port associé

Création d'un descripteur de service

À partir d'un descripteur de socket UDP :

```
SVCXPRT* svcudp_create(int socket);
```

À partir d'un descripteur de socket TCP :

```
SVCXPRT* svctcp_create(int socket, uint_t taille_buf_send,
    uint_t taille_buf_reception);
```

Destruction d'un descripteur de service

```
void svc_destroy(SVCXPRT* service);
```

Couche basse : Point de vue serveur

Enregistrer un service au près du port mapper

```
bool_t svc_register(SVCXPRT* service,  //descripteur de service
                    u_long prog,       // id programme
                    u_long vers,       // id version de programme
                    void (*dispatch)(struct svc_req* , SVCXPRT*),
                        // fonction aiguillage
                    int protocole); // IPROTO_UDP ou IPROTO_TCP
```

L'argument `dispatch`

Pointeur de fonction sur une fonction d'aiguillage appelée à chaque fois que le serveur reçoit un appel distant sur la version `vers` du programme `prog`.

la structure `svc_req`

```
struct svc_req {
 rpcprog_t              rq_prog;//num prog
 rpcvers_t              rq_vers;//num vers
 rpcproc_t              rq_proc;//num procedure
 struct opaque_auth     rq_cred;//pour l'authentification
 caddr_t                rq_clntcred;//pour l'authentification
 SVCXPRT*               *rq_xprt;//pointeur sur le descripteur associe
}
```

Couche basse : Point de vue serveur

Squelette d'une fonction d'aiguillage

```
void dispatch(struc svc* req, SVCXPRT* services){
 switch(req->rq_proc){
  case NULLPROC : //fonction 0
   ....
   return;
  case NUMPROC_1: //fonction de numero NUMPROC_1
   //traitement NUMPROC_1
   return;
  ...
  case NUMPROC_N: //fonction de numero NUMPROC_N
   //traitement NUMPROC_N
   return;
  default: //pas de fonction associee
   svcerr_noproc(service);
   return;
 }
}
```

Couche basse : Point de vue serveur

Accès aux paramètres sur la requête courante

```
bool_t svc_getargs(SVCXPRT* service,
                   xdrproc_t xdrparam,
                   void* parametre);
```

Transmettre les résultats au client

```
bool_t svc_sendreply(SVCXPRT* service,
                     xdrproc_t xdrresultat,
                     void* resultat);
```

Exemple dans dispatch :

```
case DIV:
    svc_getargs(s, xdr_Div, &div);
    //traitement division
    svc_sendreply(s, xdr_double,&rep);
    return;
```

Couche basse : Point de vue client

Le type CLIENT

Descripteur de client permettant d'identifier un client dans le protocole (ex : Machine + port client, socket de connexion serveur, ...)

Création d'un descripteur client sur UDP

```
CLIENT* clntudp_create(struct sockaddr_in * adr,
                       u_long prog,
                       u_long, vers,
                       struct timeval *delai,
                       int* socket);//socket de connexion
```

- si `adr->sin_port` est nul, consultation du portmap distant
- délai = intervalle de temps pour réémettre une requête si pas de réponse

Couche basse : Point de vue client

Création d'un descripteur de client

```
CLIENT *clnt_create(char *host,          // nom machine
                    unsigned long prog,  // id programme
                    unsigned long vers,  // id version programme
                    char *proto);        // "udp" ou "tcp"
```

Destruction d'un descripteur de service

```
void clnt_destroy(CLIENT* client);
```

Couche basse : Point de vue client

Appel de service

```
enum clnt_stat clnt_call(CLIENT* client,
                         u_long procedure,
                         xdrproc_t xdr_data,
                         char* data,
                         xdrproc_t xdrresultat,
                         char* resultat,
                         struct timeval tempsTotals);
```

Attention

temps total = temps avant abandon de la requête
mais $\neq$ du délai de garde

Appel Asynchrone

Définition

Dans un appel asynchrone le processus appelant ne se bloque pas pour attendre une réponse du serveur

Comment faire ?

Paramétrer le temps total du `clnt_call` à 0.

```
bool_t clnt_control(CLIENT* c, int requete, void* info);
```

Requête est peut être :

- `CLSET_TIMEOUT` : info pointe sur une `struct timeval` qui contient le nouveau timeout
- `CLGET_TIMEOUT` : info pointe sur une `struct timeval` étant l'actuel timeout

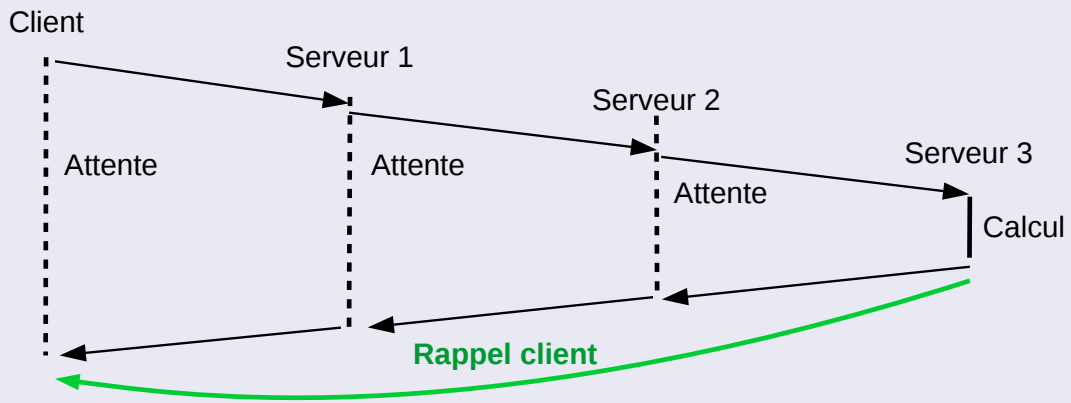
Mécanisme de rappel

Principe

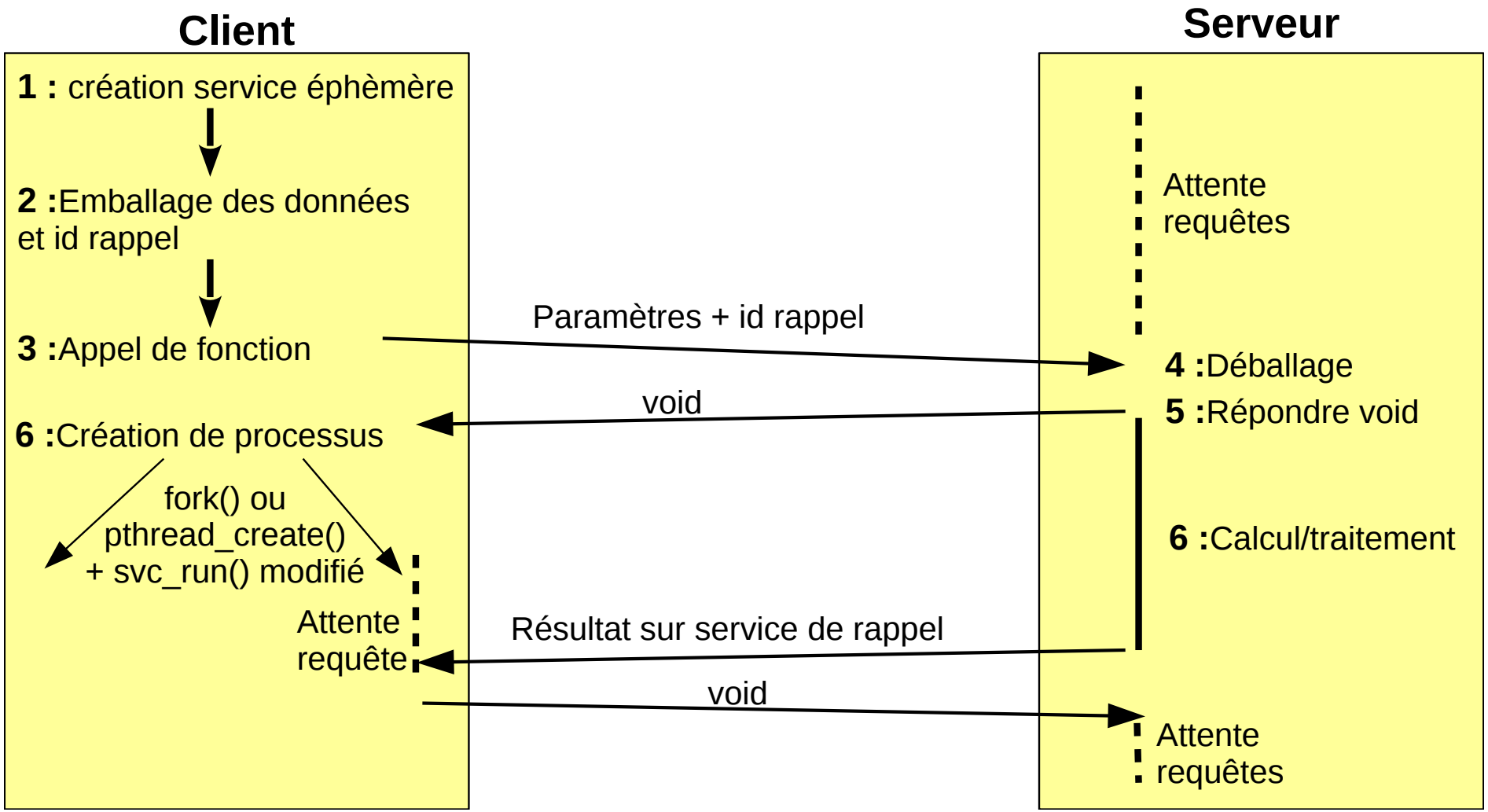
Définir sur le client un service éphémère de rappel pour qu'un serveur puisse le contacter ultérieurement

Intérêts

- Le serveur peut répondre un résultat même si le client a fait un appel asynchrone
- Un serveur S peut transférer à un autre serveur S' la requête du client
  $\Rightarrow$ S' peut contacter directement le client sans repasser par S



Mécanisme de rappel : interactions



le langage RPCL

Caractéristiques

- Permet de spécifier une interface de services dans un fichier ".x"
- Compilation = production de squelette C en couche basse :
  - talons serveur et et client
  - fonctions XDR

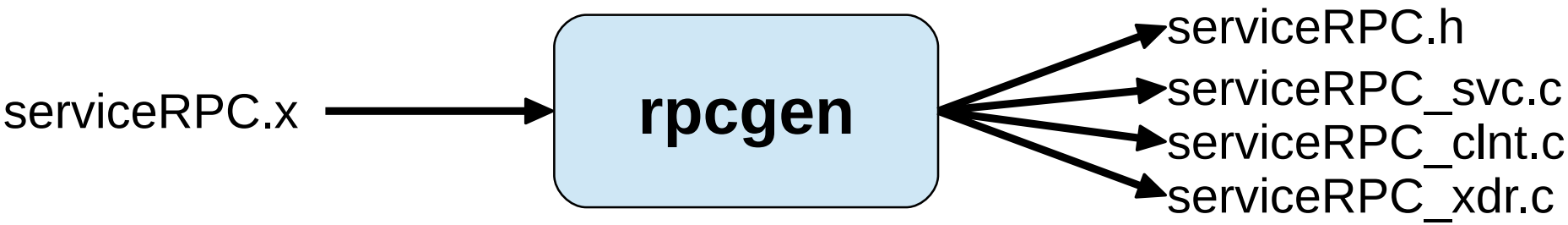
```
program NOMPROGRAM {
  version PROGVERS {
    type_retour NOMSERVICE1(type_param)=1;
    type_retour NOMSERVICE2(type_param)=2;
    ...
  }=NUMEROVERSION
} = NUMEROPROGRAM
```

Les majuscules sont conventionnelles.

RPCL : Les types

type	en RPCL	en C
vide	`void`	`void`
entier	`int, short, long`	`int, short, long`
flottant	`double, float`	`double, float`
octet	`char`	`char`
booléen	`bool`	`bool_t` (si RPC)
chaine	`string name<MAX>`	`char* name[MAX]` + \0
tableau taille fixe	`int data[SIZE]`	`int data[SIZE]`
tableau taille variable	`int data<MAXSIZE>`	`typedef struct{ int *tab; u_int len; //<= MAXSIZE }data;`
structuré	`struct coord{ int x; int y; };`	`struct coord{ int x; int y; }; typedef struct coord coord;`

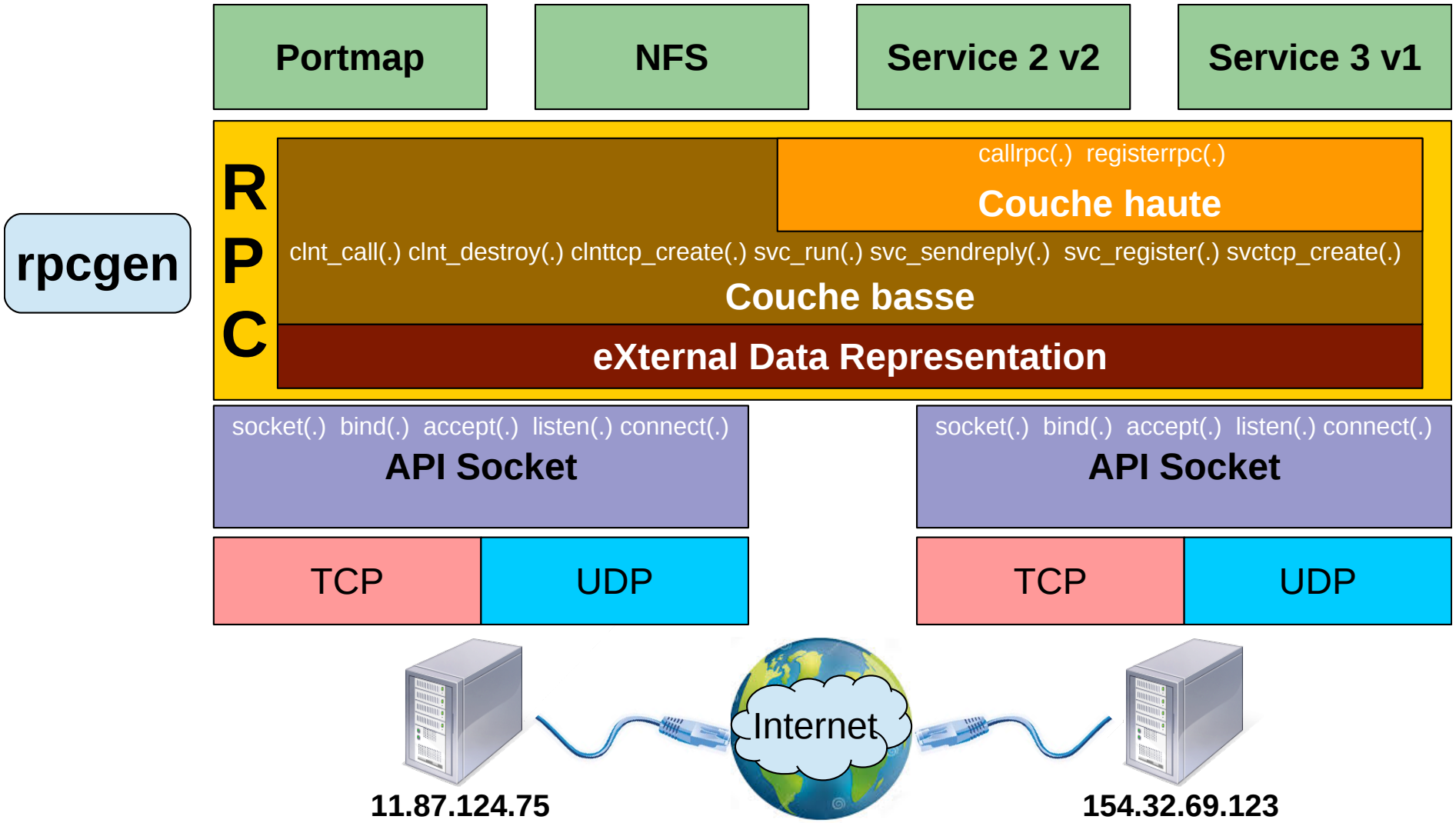
Le compilateur `rpcgen`



Principales options

Option	Interprétation
`-C`	code généré à la norme ANSI
`-a`	génère fichiers de base + exemples client/serveur + Makefile
`-M`	génère des talons *multithread-safe*
`-s type`	génère uniquement que pour le protocole type (udp ou tcp)

Synthèse


Portmap
NFS
Service 2 v2
Service 3 v1
rpcgen
RPC
callrpc(.) registerrpc(.)
Couche haute
clnt_call(.) clnt_destroy(.) clnttcp_create(.) svc_run(.) svc_sendreply(.) svc_register(.) svctcp_create(.)
Couche basse
eXternal Data Representation
socket(.) bind(.) accept(.) listen(.) connect(.)
API Socket
socket(.) bind(.) accept(.) listen(.) connect(.)
API Socket
TCP
UDP
TCP
UDP
Internet
11.87.124.75
154.32.69.123