INTRODUCTION à l'utilisation de l'appel de procédure à distance dans

ONC-RPC (Sun)

Eric Gressier

CNAM-CEDRIC

PLAN

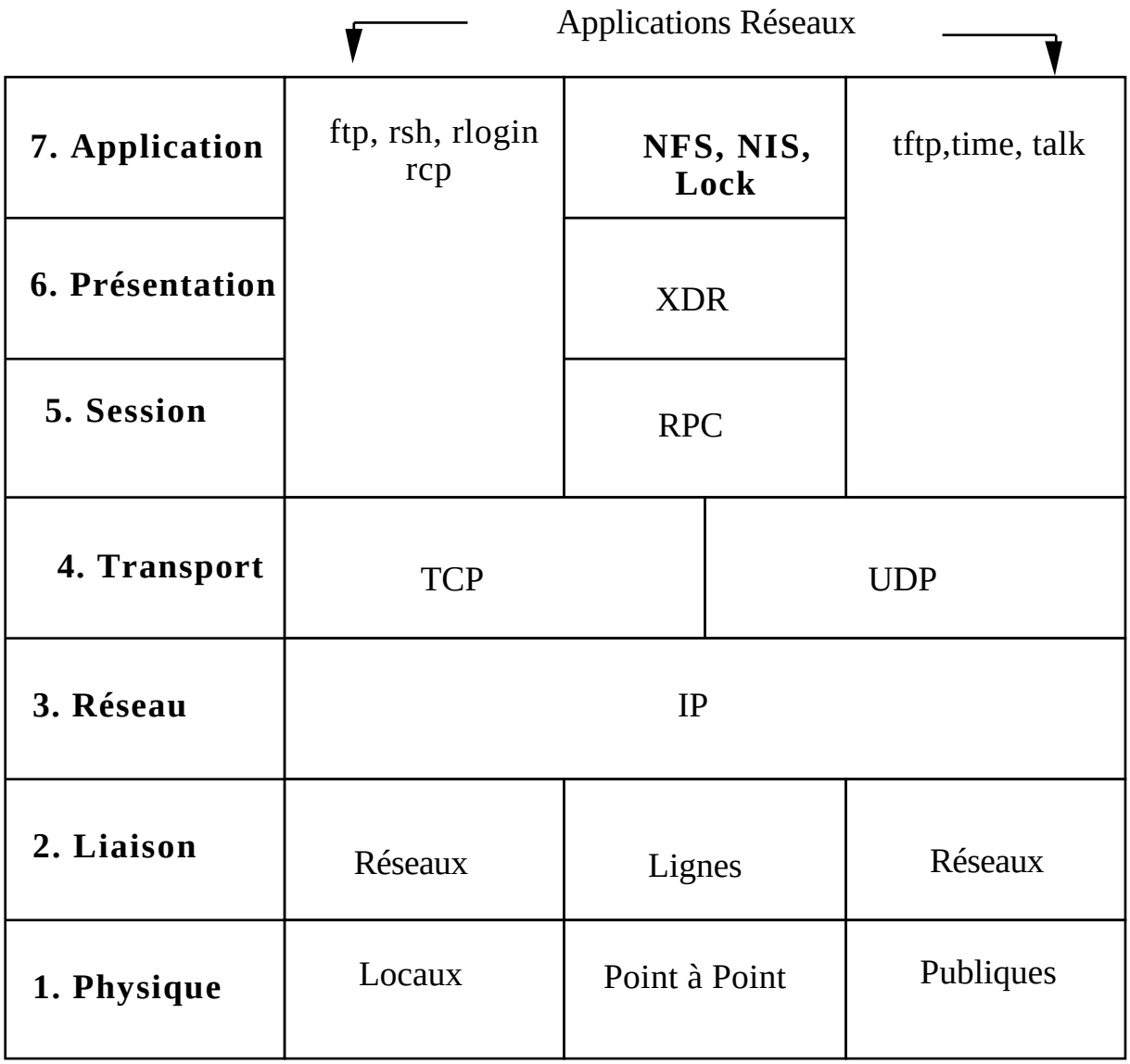
REFERENCES BIBLIOGRAPHIQUES :

Power Programming with RPC. John Bloomer. O' Reilly & Associates Inc 1992.

Internetworking with TCP/IP Volume III : Client-Server Programming and Applications, BSD socket version. Douglas E. Comer, David L. Stevens. Prentice-Hall 1993.

1. INTRODUCTION

RPC sur TCP/IP et modèle ISO

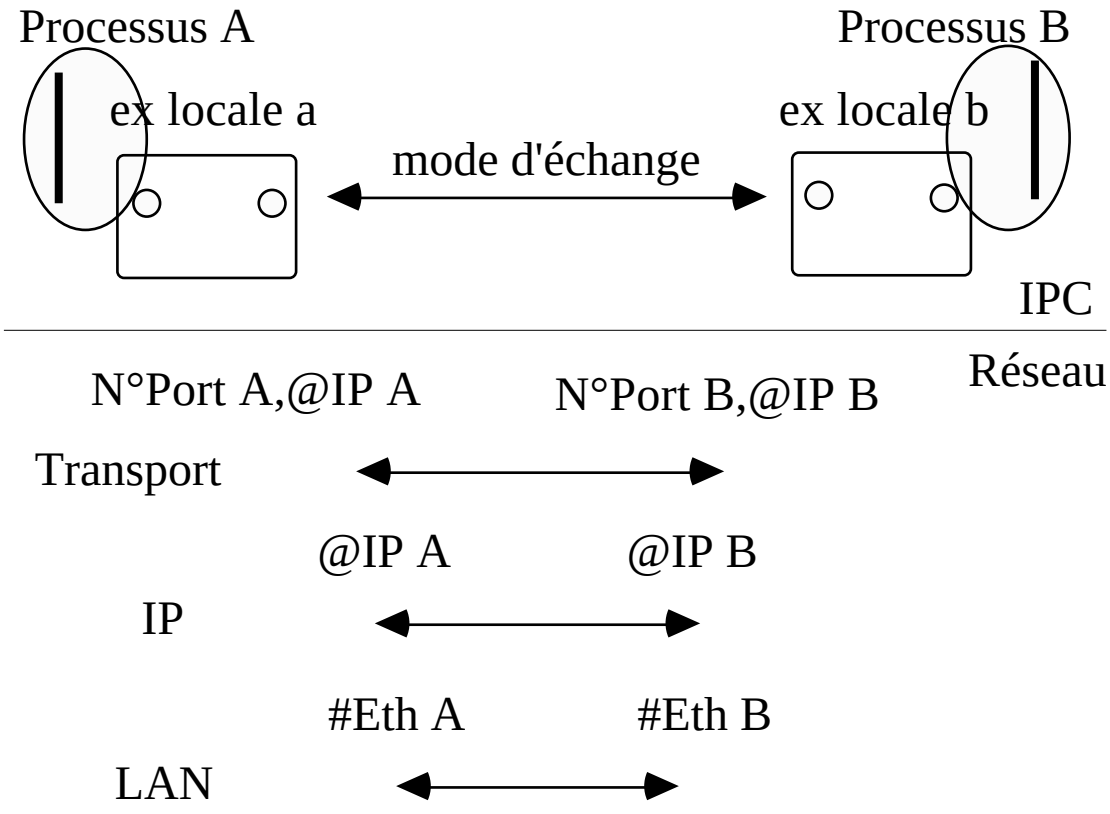


Communication IPC-socket

Association (Dialogue) IPC:

Local (un Client) Distant (un Serveur)

{processus,extrêmité,mode d'échange,extrêmité,processus}



2. Concepts de base

Concept d'Appel de Procédure à Distance

Image pour l'utilisateur :
appel de procédure dans un programme principal avec passage de paramètres

mais

à travers le réseau, de machine à machine

Nouveau Paradigme :
APD ou RPC[1]

Mécanisme pour faire de l'Exécution Répartie plus intéressant que le mode message pour le développeur d'applications

Mécanisme "naturel" pour l'écriture de clients/serveurs.

Fonctions de couche Session ou Application : contrôle d'exécution répartie

Fonctions de couche Présentation : passage de paramètres

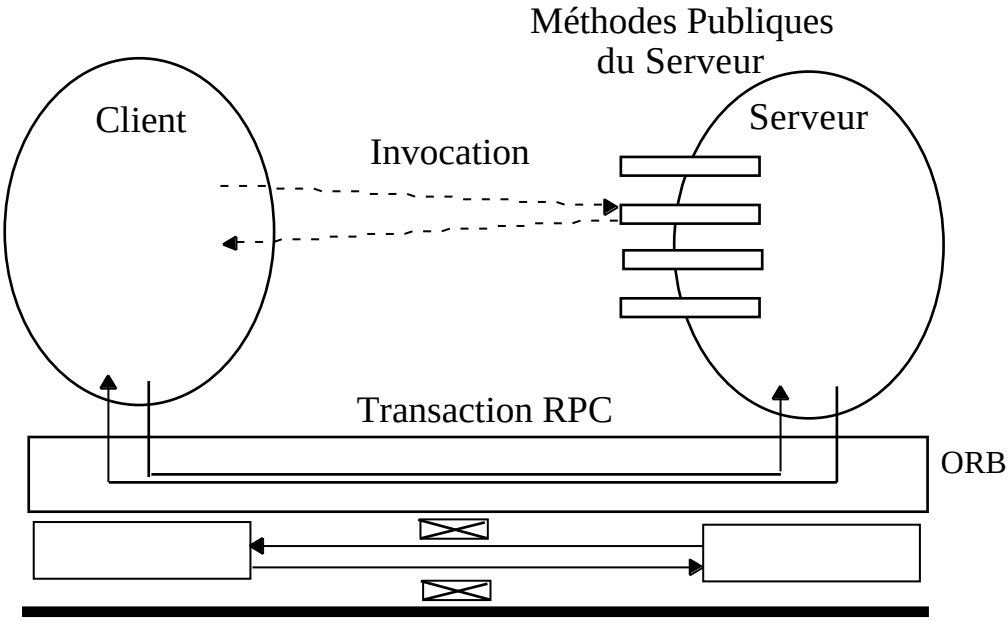
---

[1]RPC : Remote Procedure Call

RPC et OMG/CORBA

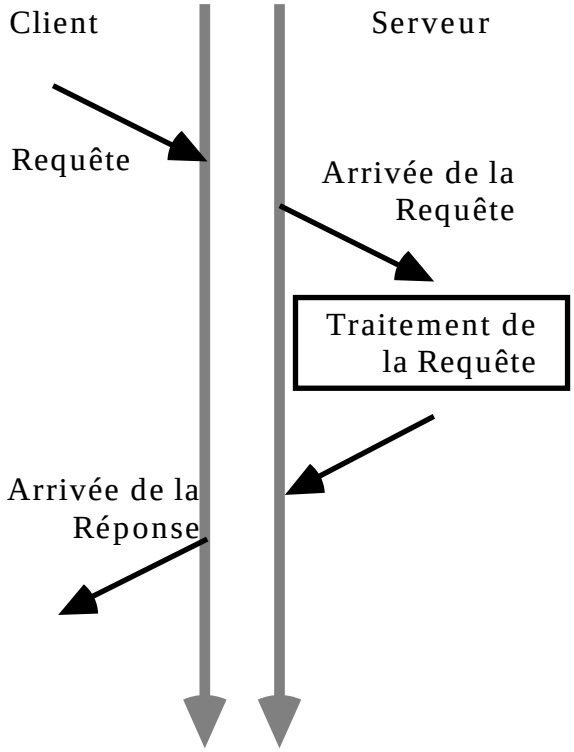
OMG/CORBA enrichit le modèle RPC.

Le Mécanisme de RPC est la base de l'invocation de méthode sur des objets distants.



Echange de Messages pour la
Transaction RPC

Principe du RPC :

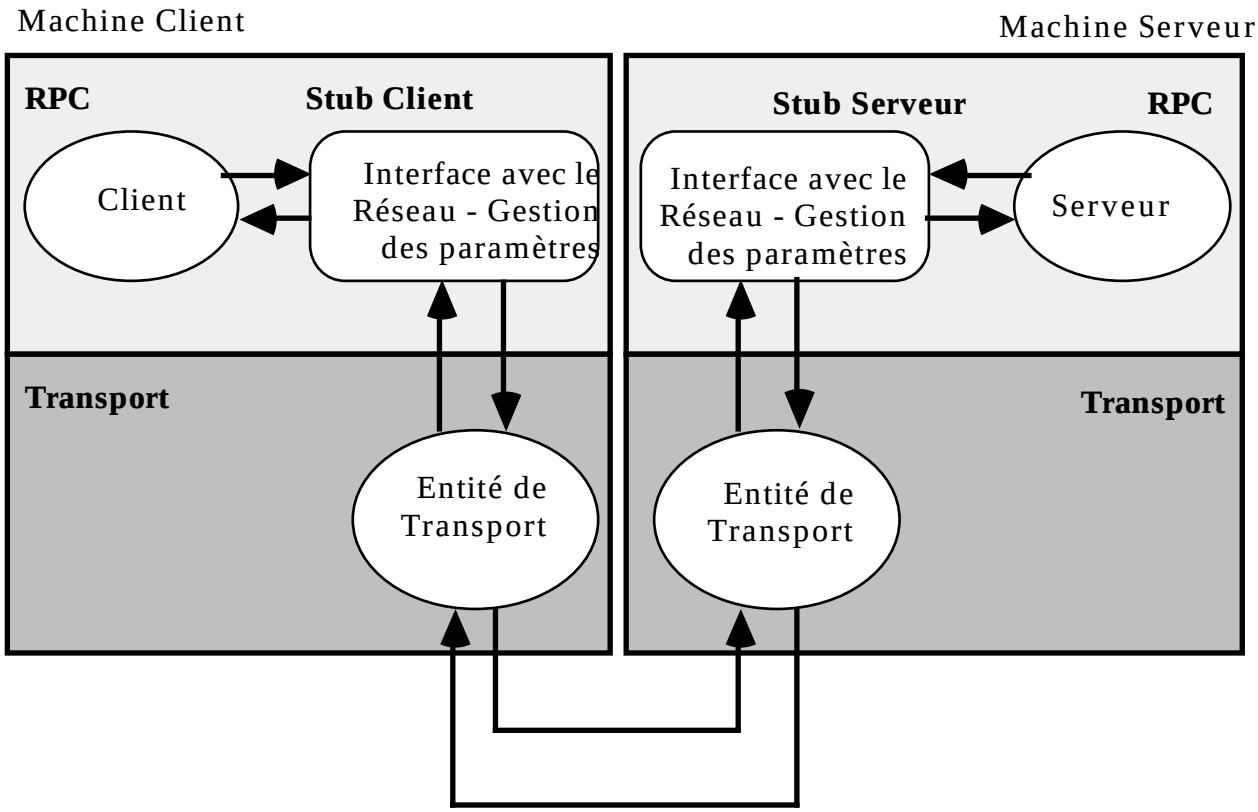


=> Problème de **panne d'un serveur**.

=> **Panne du réseau**

=> Problème de **panne d'un client** qui crée des traitements sans utilité chez le serveur : "orphelins".

Implantation du RPC :

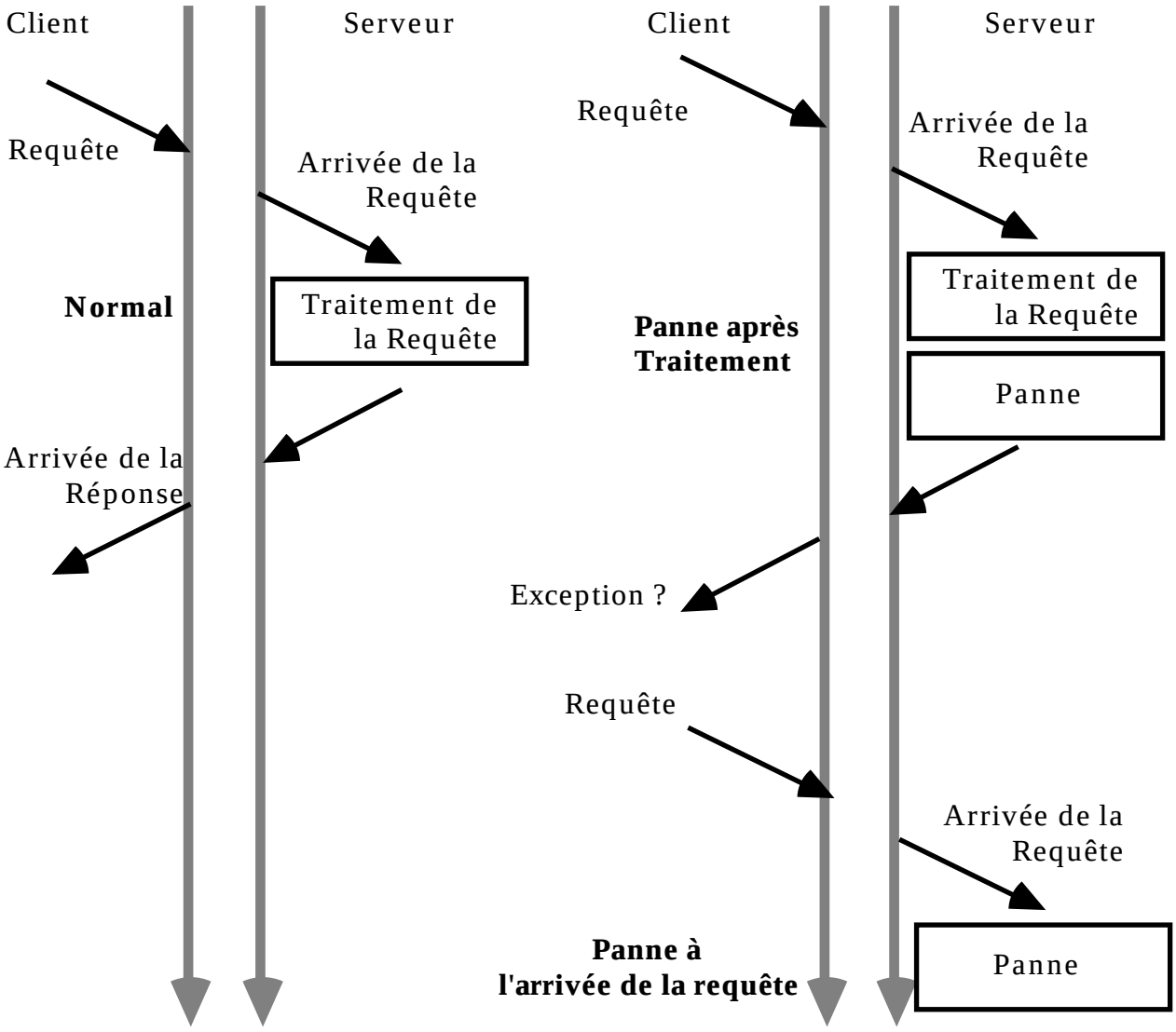


=> RPC masque le passage de paramètres par valeur

=> Pb pour le passage de paramètres par référence: un pointeur n'a plus de sens d'une machine sur l'autre, d'un espace d'adressage d'un processus à un autre, solutions :

- copy/restore,
- empêcher les passages de paramètres par référence,
- transmettre des structures de données complexes

Panne de Serveur



Sémantique du service exécuté par le serveur :

au moins une fois
au plus une fois
exactement une fois

Nature du Serveur :

avec état ou sans état

Panne du Réseau

Mode connecté ou mode datagramme

Pertes de Messages :

- **Perte de la demande :** Armer un délai de garde

- **Perte de la réponse :** Armer un délai de garde, et Problème de la sémantique d'exécution du service

idempotence des requêtes ($f^n = f$)

Le client n'a aucun moyen de savoir ce qu'a fait le serveur ...

même problème à résoudre que celui de la fermeture de connexion

Panne de Client (1)

Eliminer les traitements qui ne correspondent plus à un client demandeur (traitements orphelins)

Extermination :

Avant appel, le client **enregistre dans un journal sur disque** le nom du serveur et l'action qu'il demande.
Après la panne, il reprend le journal pour demander la destruction des traitements en cours.
Inefficace : ne résoud pas le problème des orphelins d'orphelins, ne résiste pas au partitionnement de réseau.

Réincarnation :

On divise le temps en **époques numérotées**, quand une machine redémarre, elle diffuse le fait qu'elle débute une nouvelle époque, ce qui permet aux serveurs de détruire les traitements inutiles à la réception du message diffusé.
Si le réseau est partionné, il reste des orphelins, mais ils sont détectés lors de la réponse par un numéro d'époque invalide.

Panne de Client (2)

Reincarnation douce :

Réincarnation, où le traitement n'est détruit que si le programme (et non la machine) qui a lancé le traitement a disparu.

Expiration :

Une requête dispose d'un certain délai pour s'exécuter. Si ce délai est insuffisant, le serveur doit redemander un nouveau quantum. Trouver la valeur du délai qui est très différente en fonction des requêtes.

Détruire un orphelin n'est pas simple :

- il détient des **ressources systèmes** (exemple verrous de fichiers) qui peuvent rester indéfiniment **mobilisées**
- ils peuvent avoir lancer d'autres traitements

Edition de liens

Localiser la procédure à exécuter
=
Trouver où est le serveur ?

1. **Edition de liens statique :** L'adresse du serveur est écrite "en dur" dans le code du client.
Difficile en cas de changement : si le serveur migre, s'il change d'adresse, ...

2. **Edition de liens dynamique :** L'adresse n'est connue qu'au moment de l'invocation de la procédure
a. Le serveur **exporte son interface** en envoyant la spécification de ses procédures à un serveur dédié à cette fonction (éditeur de liens) : serveur centralisé ou service réparti. Il est enregistré et connu.
Le serveur donne une référence sur lui-même (adresse réseau, nom unique ...)
b. Le Client s'adresse à l'**éditeur de liens**, pour connaître le serveur qui sert la procédure dont il demande l'exécution. L'éditeur de lien lui donne l'adresse d'un serveur disponible (ils peuvent être plusieurs à pouvoir fournir le même service -> tolérance aux pannes)
c. Le client s'adresse au serveur pour exécuter la procédure souhaitée

3. ONC-RPC

RPC Sun

Adressage des serveurs/des procédures

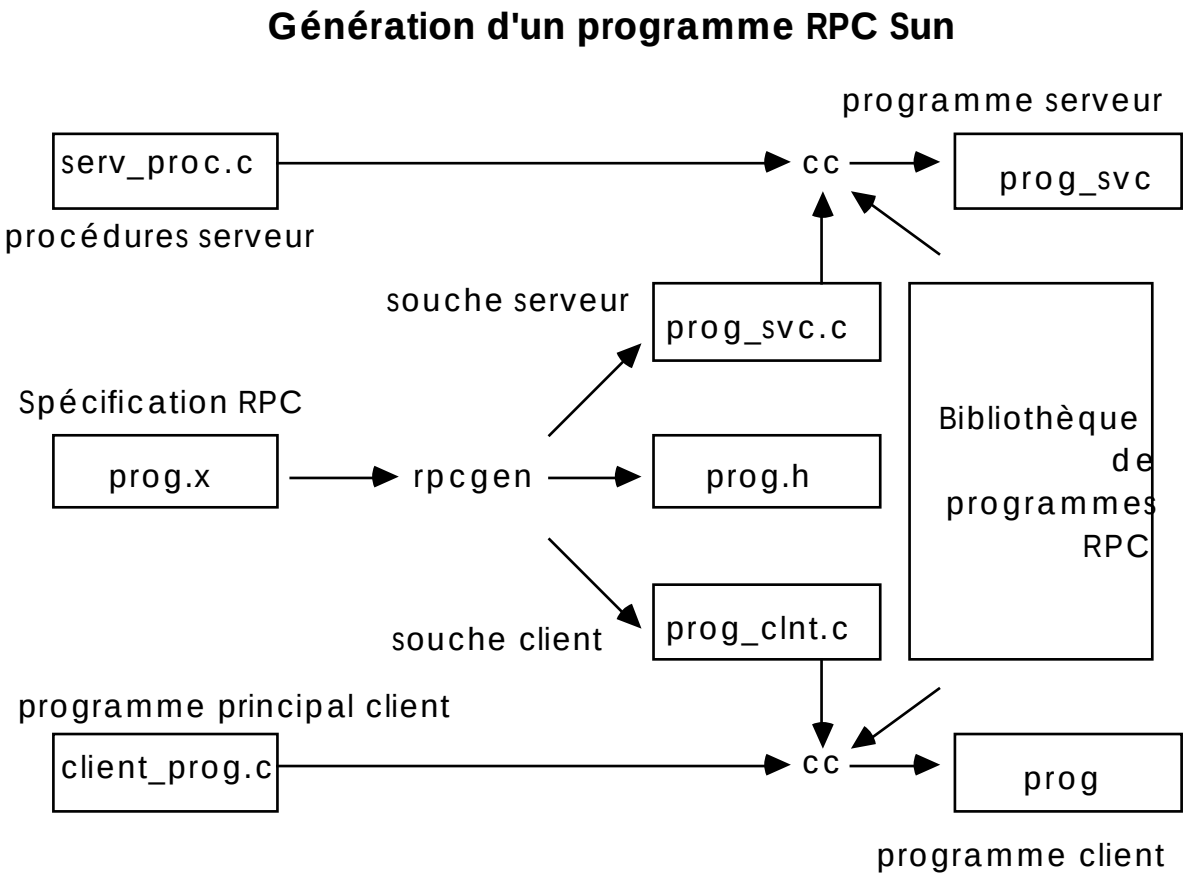
Il manipule des "N°de Programme" et des "N°de Version" (Sun)

=> Il faut gérer une correspondance avec les *N°de Port*

Interface plus souple pour les programmes
=>
utilisation de rpcgen (SUN), NIDL (Apollo)

ONC vs NCS-DCE

Environnement de Génération de RPC



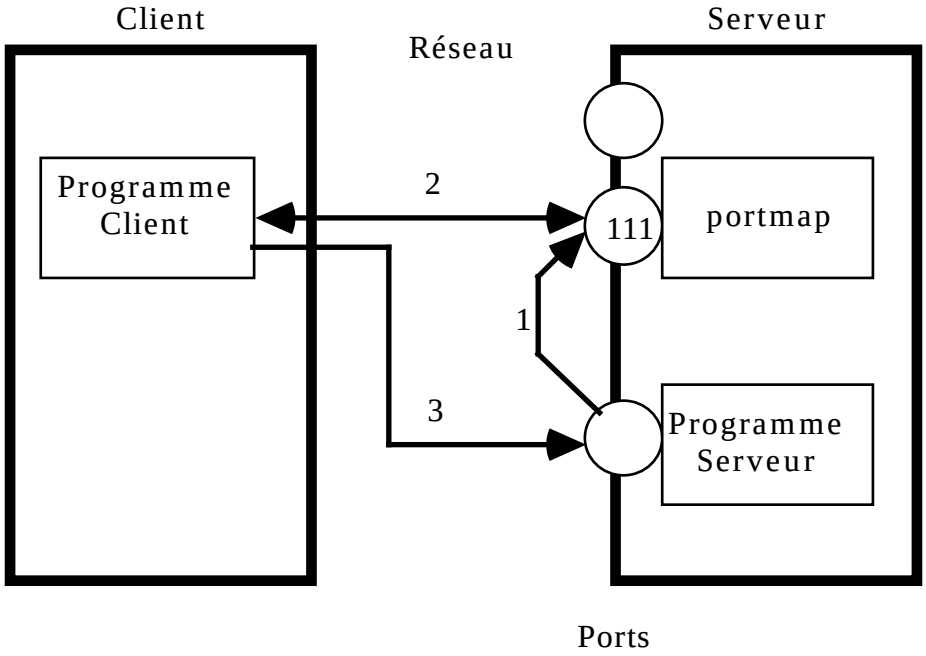
On retrouve le même type d'environnement pour le développement d'application avec CORBA.

Sémantique du RPC Sun :

au moins une fois

Localisation d'un serveur : *portmapper*

Il est appelé par tout client et retourne le N°Port affecté au service qui est demandé.



1. le programme serveur fait enregistrer son numéro de port, son numéro de programme, et son numéro de version.

2. le programme client interroge le daemon "portmap" du serveur, et obtient le numéro de port associé au programme serveur,

3. le programme client appelle le programme serveur.

Interrogation du portmapper

Table de correspondance gérée par le programme "portmap" obtenue par la commande rpcinfo -p[2] :

program	vers	proto	port	service
100000	2	tcp	111	portmap
100000	2	udp	111	portmap
…	…	…	…	…
100004	2	udp	655	ypserv
100004	2	tcp	656	ypserv
100004	1	udp	655	ypserv
100004	1	tcp	656	ypserv
…	…	…	…	…
102999	10	udp	7000	monserveur
…	…	…	…	…

[2]Quand NFS ne fonctionne pas, vérifier que portmap s'exécute bien :

ps -aux I grep portmap

Présentation des données

Problèmes Posés par l'échange de données entre machines hétérogènes :

- la longueur de représentation des données
  processeur 16 bits ,32 bits, 64 bits

- la représentation interne des données en mémoire

- le cadrage des données dans les enregistrements:

sur des multiples de deux octets (micro), quatre octets (+répandu) ou huit octets (cray) ???

Solution Sun :

eXternal Data Représentation : XDR
(différente de la solution ASN.1-BER de l'ISO)

- Fournir un ensemble de types prédéfinis voisins de ceux qui existent dans le langage C ou le Pascal :

=> types de base

* entier, entier non signé,
* entier long, entier long non signé,
* réel, réel double précision,
* booléen,
* énumération,
* suite d'octets de longueur fixe ou variable (données opaques -> file handle de NFS), chaine de caractères

=> types composés

* tableau de longueur fixe ou variable,
* structure, union,

- mécanisme qui permet la construction de types "privés":

* définition de constantes (*#define*),
* définitions de types (*typedef*),
* type à interprétation multiple (à partir d'un mécanisme inspiré du *switch* en programmation)

Hypothèses faites dans XDR :

- l'octet (8 bits significatifs) est l'unité de base reconnue par toutes les machines,

- la représentation big endian est adoptée

- alignement des blocs de données sur un multiple de 4
=> un bloc de données peut être suivi de 0 à 3 octets null suivant sa taille,

- une bibliothèque de fonctions qui servent aussi bien à encoder et qu'à décoder.

Exemple : fonction d'encodage et de décodage d'un réel double précision

bool_t xdr_double(xdrs, dp)
XDR *xdrs ;
double *dp;

structures de données annexes :

bool_t : TRUE, FALSE

enum xdr_op { XDR_ENCODE=0, XDR_DECODE=1, XDR_FREE=2 }

```
typedef struct {
        enum xdr_op       x_op ;
        struct        xdr_ops       {
                bool_t        (*x_getlong) (); /* get long from stream */
                bool_t        (*x_putlong) (); /* put long to stream */
                bool_t        (*x_getbytes) (); /* get bytes from stream */
                bool_t        (*x_putbytes) (); /* putbytes to stream */
                u_int         (*x_getpostn) (); /*return stream offset */
                bool_t        (*x_setpostn) (); /* reposition offset */
                caddr_t       (*x_inline) (); /* ptr to buffered data */
                VOID(*x_destroy) (); /*free private area */
        } *x_ops;
        caddr_t       x_public;  /* users' data */
        caddr_t       x_private;  /* pointer to private data */
        caddr_t       x_base;  /*private for position info */
        int           x_handy;  /* extra private word */
} XDR;
```

Exemple d'utilisation de RPCGEN et de mise en oeuvre du RPC Sun

Application de consultation d'un dictionnaire[3] :

Opérations de base :

Commande (1 caractère)	Argument	Effet
I	-aucun-	Initialise le Dictionnaire en le vidant
i	mot	Insèrer un mot dans le dictionnaire
d	mot	Enlever un mot dans le dictionnaire
l	mot	Chercher un mot dans le dictionnaire
q	-aucun-	Quitter

[3]Exemple tiré de : "Internetworking with TCP/IP Volume III : Client-Server Programming and Applications. BSD Socket Version". Douglas E. Comer, David L. stevens. Prentice Hall. 1993.

Application version centralisée - Appel de Procédure Local

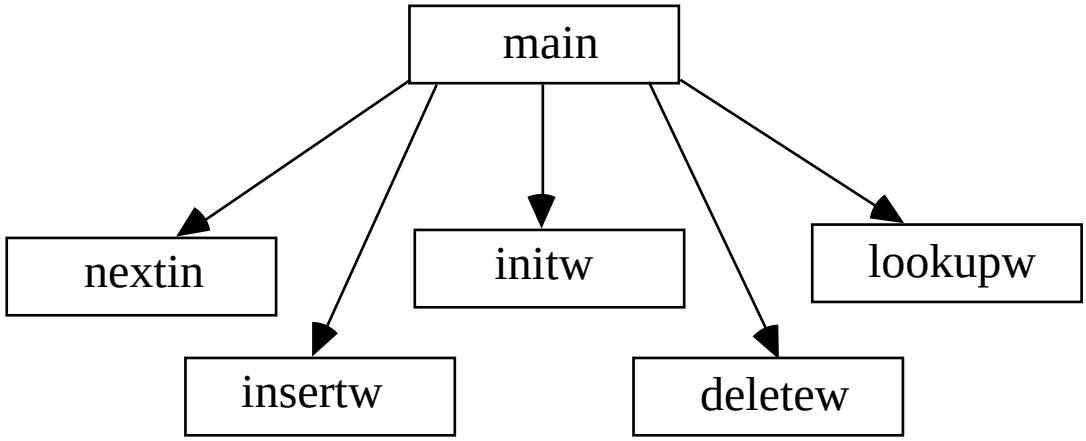
```
/* dict.c - main, initw, nextin, insertw, deletew, lookupw */
#include <stdio.h>
#include <ctype.h>

#define     MAXWORD     50          /* maximum length of a command or word      */
#define DICTSIZ 100                 /* maximum number of entries in diction.*/
char dict[DICTSIZ][MAXWORD+1];      /* storage for a dictionary of words        */
int     nwords = 0;                 /* number of words in the dictionary        */

/*---------------------------------------------------------------------
 * main - insert, delete, or lookup words in a dictionary as specified
 *--------------------------------------------------------------------- */
void main(argc, argv)
int     argc; char *argv[];
{
        char  word[MAXWORD+1]; /* space to hold word from input line    */
        char  cmd;
        int           wrdlen;                   /* length of input word                 */
        while (1) {
                wrdlen = nextin(&cmd, word);
                if (wrdlen < 0)     exit(0);
                switch (cmd) {
                case 'I':   initw();                    /* "initialize"     */
                        printf("Dictionary initialized to empty.\n");
                        break;
                case 'i':   insertw(word);              /* "insert" */
                        printf("%s inserted.\n",word);
                        break;
                case 'd':   if (deletew(word))          /* "delete"         */
                                printf("%s deleted.\n",word);
                        else   printf("%s not found.\n",word);
                        break;
                case 'l':   if (lookupw(word))          /* "lookup"         */
                                printf("%s was found.\n",word);
                        else   printf("%s was not found.\n",word);
                        break;
                case 'q':                               /* quit             */
                        printf("program quits.\n");
                        exit(0);
                default:                                /* illegal input    */
                        printf("command %c invalid.\n", cmd);
                        break;
                }
        }
}
```

```
/*-------------------------------------------------------------------------
 * nextin - read a command and (possibly) a word from the next input line
 *-------------------------------------------------------------------------*/
int nextin(cmd, word)

char  *cmd, *word;
{
        int     i, ch;

        ch = getc(stdin);
        while (ch == ' ')  ch = getc(stdin);
        if (ch == EOF)              return -1;
        *cmd = (char) ch;
        ch = getc(stdin);
        while (ch == ' ')           ch = getc(stdin);
        if (ch == EOF)              return -1;
        if (ch == '\n')             return 0;
        i = 0;
        while (ch != ' ' && ch != '\n') {
                if (++i > MAXWORD-1) {
                        printf("error: word too long.\n");
                        exit(1);
                }
                *word++ = ch;
                ch = getc(stdin);
        }
        *word = '\0';
        return i;
}

/*-------------------------------------------------------------------------
 * initw - initialize the dictionary to contain no words at all
 *-------------------------------------------------------------------------*/
int initw()

{
        nwords = 0;
        return 1;
}

/*-------------------------------------------------------------------------
 * insertw - insert  a word in the dictionary at the end of it
 *-------------------------------------------------------------------------*/
int insertw(word)

char  *word;
{
        strcpy(dict[nwords], word);
        nwords++;
        return nwords;
}
```

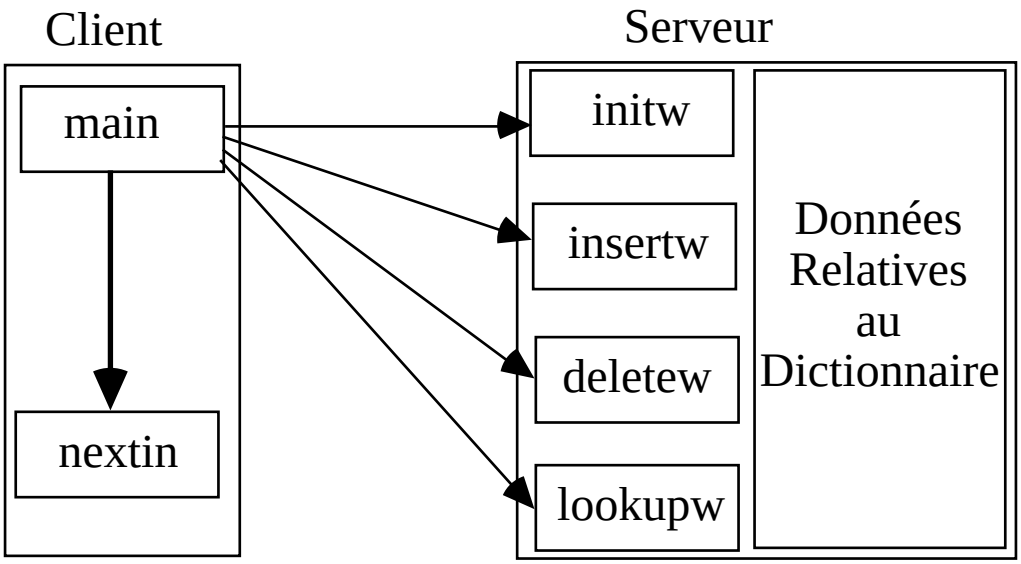
```
/*-------------------------------------------------------------------------
 * deletew - delete  a word from the dictionary
 *-------------------------------------------------------------------------*/
int deletew(word)

char  *word;
{
        int     i;

        for (i=0 ; i<nwords ; i++)
                if (strcmp(word, dict[i]) == 0) {
                        nwords--;
                        strcpy(dict[i], dict[nwords]);
                        return 1;
                }
        return 0;
}

/*-------------------------------------------------------------------------
 * lookupw - look up a word in the dictionary
 *-------------------------------------------------------------------------*/
int lookupw(word)

char  *word;
{
        int     i;

        for (i=0 ; i<nwords ; i++)
                if (strcmp(word, dict[i]) == 0)
                        return 1;
        return 0;
}
```

Découpage du programme en un Client et un Serveur

Graphe d'Appel à partir du programme principal (main) :



Client et Serveur :



Remarque : On a un découpage qui ressemble à un objet et à ses méthodes.

Un Programme Client : dict1.c

```
/* dict1.c - main, nextin */
#include <stdio.h>
#include <ctype.h>
#defineMAXWORD   50              /* maximum length of a command or word     */

/*------------------------------------------------------------------------
 * main - insert, delete, or lookup words in a dictionary as specified
 *------------------------------------------------------------------------*/
int main(argc, argv)
int     argc;
char    *argv[];
{
        char    word[MAXWORD+1];        /* space to hold word from input line      */
        char    cmd;
        int     wrdlen;                 /* length of input word                    */

        while (1) {
                …
                switch (cmd) {
                case 'I':       initw()                 /* "initialize" */
                        …
                        break;
                case 'i':       insertw(word);          /* "insert" */
                        …
                        break;
                case 'd':       if (deletew(word))      /* "delete" */
                        …
                        break;
                case 'l':       if (lookupw(word))      /* "lookup" */
                        …
                        break;
                case 'q':       /* quit */
                        …
                        exit(0);
                default:        /* illegal input */
                        …
                        break;
                }
        }
}
/*------------------------------------------------------------------------
 * nextin - read a command and (possibly) a word from the next input line
 *------------------------------------------------------------------------*/
int nextin(cmd, word)
char    *cmd, *word;
{
…
}
```

Un Programme Serveur : dict2.c

```
/* dict2.c - initw, insertw, deletew, lookupw */

#define MAXWORD   50                    /* maximum length of a command or word   */
#define DICTSIZ 100                     /* maximum number of entries in diction. */
char    dict[DICTSIZ][MAXWORD+1];       /* storage for a dictionary of words     */
int     nwords = 0;                     /* number of words in the dictionary     */

/*------------------------------------------------------------------------
 * initw - initialize the dictionary to contain no words at all
 *------------------------------------------------------------------------*/
int initw()
{
…
}

/*------------------------------------------------------------------------
 * insertw - insert  a word in the dictionary
 *------------------------------------------------------------------------*/
int insertw(word)
char    *word;
{
…
}

/*------------------------------------------------------------------------
 * deletew - delete  a word from the dictionary
 *------------------------------------------------------------------------*/
int deletew(word)
char    *word;
{
…
}

/*------------------------------------------------------------------------
 * lookupw - look up a word in the dictionary
 *------------------------------------------------------------------------*/
int lookupw(word)
char    *word;
{
…
}
```

Etape 1 : Spécification pour le Générateur RPCGEN

Spécification des procédures :

. constantes communes au client et au serveur

. déclaration des types de données utilisées

. déclaration des procédures du serveur et des paramètres d'appel

```
/* dict.x - RPC declarations for dictionary program, */
const  MAXWORD = 50 [4];      /* maximum length of a command or word */
const  DICTSIZ = 100;         /* number of entries in dictionary     */

struct example {              /* unused structure declared here to   */
        int     exfield1;     /* illustrate how rpcgen builds XDR    */
        char    exfield2;     /* routines to convert structures.     */
};

/*---------------------------------------------------------------------
 * RDICTPROG - remote program that provides initw, insert, delete, and lookup
 *---------------------------------------------------------------------*/

program RDICTPROG {                 /* name of remote program (not used)*/
   version RDICTVERS {              /* declaration of version (see below) */
        int INITW(void)         = 1;    /* first procedure in this program    */
        int INSERTW(string)     = 2;    /* second procedure in this program   */
        int DELETEW(string)     = 3;    /* third procedure in this program    */
        int LOOKUPW(string)     = 4;    /* fourth procedure in this program   */
   } = 1;                       /* definition of the program version          */
} = 0x30090949;                 /* remote program number (must be unique)     */
```

rpcgen rdict.x

---

[4]Ce qui change en passant du programme version locale en RPC : MAXWORD était déclaré par

#define MAXWORD 50

Procédures de conversion XDR Résultant de RPCGEN : rdict_xdr.c

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

bool_t
xdr_example(xdrs, objp)
        XDR *xdrs;
        example *objp;
{
        if (!xdr_int(xdrs, &objp->exfield1)) {
                return (FALSE);
        }
        if (!xdr_char(xdrs, &objp->exfield2)) {
                return (FALSE);
        }
        return (TRUE);
}
```

RPCGEN produit un fichier rdict.h qu'il faudra inclure dans certains fichiers de code source.

Fonctions de conversion utilisées :

`xdr_void()` filtre pour coder/décoder le type C void, utilisé avec les procédure qui n'ont pas de paramètres en entrée ou en sortie

`xdr_int()` filtre pour coder/décoder le type C int, utilisé pour convertir des entiers

`xdr_wrapstring()` filtre pour coder/décoder les chaines de caractères C terminées par NULL et de longueur variable

pour libérer la place allouée dans la manipulation des variables dans les souches, il faut utiliser la procédure `xdr_free()`

Programmation avec les appels RPC haut niveau - côté serveur

Pour ce mode de programmation, on fait l'hypothèse qu'on utilise le protocole UDP.

Primitives Côté serveur :

registerrpc()

```
int registerrpc(prognum,
                versnum,
                   procnum,
                      procname,
                         inproc,
                            outproc)

    u_long prognum,v ersum, procnum;
    char *(*procname)();
    xdrproc_t inproc, outproc;
```

enregistrement d'une procédure au niveau du portmapper, les procédures inproc et outproc sont des filtres de conversion, et procnam est la procédure déclarée sur le serveur

svc_run()

```
void   svc_run();
```

lance le serveur, et le met en attente de requêtes

Programmation avec les appels RPC haut niveau - côté client

Primitive Côté client :

callrpc()

```
int callrpc(host,
            prognum,
               versnum,
                  procnum,
                     inproc,
                        in,
                           outproc,
                              out)

    char *host;
    u_long prognum, versnum, procnum;
    char *in, *out;
    xdr_proct inproc, outproc;
```

appel de la procédure distante effectif

L'utilisation de l'interface de haut niveau implique :
- on utilise UDP (socket en mode DGRAM)
- la taille des paramètres ne peut excéder 8Ko
- à chaque appel de callrpc, il y a une édition de lien complète avec la procédure demandée (interrogation de portmap avant d'appeler la procédure), d'où un manque de performance
- le serveur fonctionne sur le mode "serveur itératif" (une requête à la fois
- l'interface RPC coté serveur gère l'aiguillage vers les procédures qu'implante le programme serveur

Coté Serveur : rdict2.c (utilise dict2.c)

```
/* rdict2.c - initw, insertw, deletew, lookupw */
#include <rpc/rpc.h>
#include "rdict.h"
/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

extern char     dict[];         /* storage for a dictionary of words    */
extern int      nwords;         /* number of words in the dictionary   */

main()
{

extern int initw();
extern int insertw();
extern int deletew();
extern int lookupw();

registerrpc(RDICTPROG,RDICTVERS,INITW,initw,xdr_int,xdr_int);
registerrpc(RDICTPROG,RDICTVERS,INSERTW,insertw,xdr_wrapstring,xdr_i
nt);
registerrpc(RDICTPROG,RDICTVERS,DELETEW,deletew,xdr_wrapstring,xdr
_int);
registerrpc(RDICTPROG,RDICTVERS,LOOKUPW,lookupw,xdr_wrapstring,xdr
_int);


svc_run();

}
```

le code serveur est généré par :
cc -o rdict2 rdict_xdr.c rdict2.c dict2.c

le lancement du serveur se fait par :
rdict2 &

Coté Client : rdict1.c

```
/* rdict1.c - main, nextin */
#include <stdio.h>
#include <ctype.h>
#include <rpc/rpc.h>
#include "rdict.h"

#defineMAXWORD   50              /* maximum length of a command or word     */

/*---------------------------------------------------------------------
 * main - insert, delete, or lookup words in a dictionary as specified
 *---------------------------------------------------------------------*/
int main(argc, argv)
int     argc;
char    *argv[];
{
        char    word[MAXWORD+1];        /* space to hold word from input line      */
        char    cmd;
        int     wrdlen;                 /* length of input word                    */

        while (1) {
                …
                switch (cmd) {
                case 'I':       initw()                 /* "initialize" */
                        …
                        break;
                case 'i':       insertw(word);          /* "insert" */
                        …
                        break;
                case 'd':       if (deletew(word))      /* "delete" */
                        …
                        break;
                case 'l':       if (lookupw(word))      /* "lookup" */
                        …
                        break;
                case 'q':       /* quit */
                        …
                        exit(0);
                default:        /* illegal input */
                        …
                        break;
                }
        }
}
/*---------------------------------------------------------------------
 * nextin - read a command and (possibly) a word from the next input line
 *---------------------------------------------------------------------*/
int nextin(cmd, word)
char    *cmd, *word;
{
…
```

```
}
```

```
/*-----------------------------------------------------------------------
 * initw - initialize the dictionary to contain no words at all
 *-----------------------------------------------------------------------*/
int initw()

{
        extern bool_t xdr_int();

        enum clnt_stat clnt_stat;

        int n;

        clnt_stat = callrpc("serveur", RDICTPROG, RDICTVERS, INITW,
                                                        xdr_int, &i,
                                                                xdr_int,
&n);
        if (clnt_stat != 0) {
                clnt_perrno(clnt_stat);
        } else          {
                        return n;
        }

}

/*-----------------------------------------------------------------------
 * insertw - insert  a word in the dictionary at the end of it
 *-----------------------------------------------------------------------*/
int insertw(word)

char  *word;
{
        extern bool_t xdr_wrapstring();
        extern bool_t xdr_int();

        enum clnt_stat clnt_stat;

        int n;

        clnt_stat = callrpc("serveur", RDICTPROG, RDICTVERS,
INSERTW,
                                                xdr_wrapstring, word,
                                                        xdr_int,
&n);
        if (clnt_stat != 0) {
                clnt_perrno(clnt_stat);
        } else          {
                        return n;
        }

}
```

```
/*-----------------------------------------------------------------------
 * deletew - delete  a word from the dictionary
 *-----------------------------------------------------------------------*/
int deletew(word)

char  *word;
{
        extern bool_t xdr_wrapstring();
        extern bool_t xdr_int();

        enum clnt_stat clnt_stat;

        int n;

      clnt_stat = callrpc("serveur", RDICTPROG, RDICTVERS,
DELETEW,
                                                xdr_wrapstring, word,
                                                        xdr_int,
&n);
        if (clnt_stat != 0) {
                clnt_perrno(clnt_stat);
        } else          {
                        return n;
        }

}

/*-----------------------------------------------------------------------
 * lookupw - look up a word in the dictionary
 *-----------------------------------------------------------------------*/
int lookupw(word)

char  *word;
{
        extern bool_t xdr_wrapstring();
        extern bool_t xdr_int();

        enum clnt_stat clnt_stat;

        int n;

      clnt_stat = callrpc("serveur", RDICTPROG, RDICTVERS,
LOOKUPW,
                                                xdr_wrapstring, word,
                                                        xdr_int,
&n);
        if (clnt_stat != 0) {
                clnt_perrno(clnt_stat);
        } else          {
                        return n;
        }
```

```
}
```

le code client est généré par :
cc -o rdict1 rdict_xdr.c rdict1.c

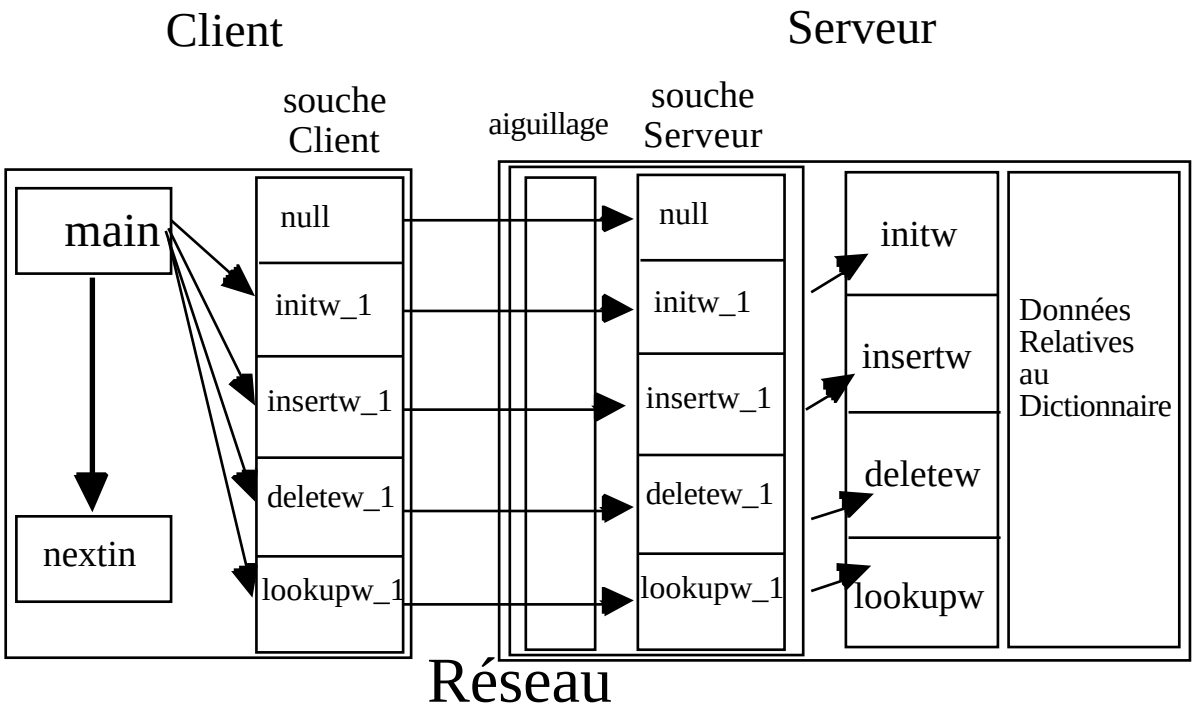
Programmation avec les appels RPC bas niveau

Le Fichier des types de données : rdict.h donne qq informations supplémentaires :

```
#define MAXWORD 50
#define DICTSIZ 100

struct example {
        int exfield1;
        char exfield2;
};

typedef struct example    example;
bool_t                    xdr_example();

#define RDICTPROG    ((u_long)0x30090949)
#define RDICTVERS    ((u_long)1)

#define INITW        ((u_long)1)
#define INSERTW      ((u_long)2)
#define DELETEW      ((u_long)3)
#define LOOKUPW      ((u_long)4)

extern int *initw_1();
extern int *insertw_1();
extern int *deletew_1();
```

```
extern int *lookupw_1();
```

Relations souches-programmes



Procédure RPC clnt_call()

```
enum clnt_stat
clnt_call(clnt, procnum,inproc, in, outproc, out, timeout)
    CLNT * clnt;
    u_long procnum;
    xdrproc_t inproc, outproc;
    char *in, *out;
    struct timeval timeout ;
```

clnt_call provoque l'appel au serveur :

- inproc sert à encoder, et outproc à décoder, ce sont des procédures filtres XDR.
- CLNT est une poignée (handle) qui identifie la procédure sur le serveur de n° procnum -> il faudra le récupérer !
- *in paramètres d'appel et *out paramètres résultat

rappel :

```
struct timeval {
    long tv_sec; /* secondes */
    long tv_usec;/* micro-secondes */
};
```

Souche Client produite par RPCGEN : rdict_clnt.c

```
#include <rpc/rpc.h>
#include <sys/time.h>
#include "rdict.h"
/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

int *initw_1(argp, clnt)
        void *argp; CLIENT *clnt;
{
static int res;
bzero((char *)&res, sizeof(res));
if (clnt_call(clnt, INITW, xdr_void, argp, xdr_int, &res, TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
return (&res);
}

int *insertw_1(argp, clnt)
        char **argp; CLIENT *clnt;
{
static int res;
bzero((char *)&res, sizeof(res));
if (clnt_call(clnt,INSERTW,xdr_wrapstring,argp,xdr_int,&res,TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
return (&res);
}

int *deletew_1(argp, clnt)
        char **argp; CLIENT *clnt;
{
static int res;
bzero((char *)&res, sizeof(res));
if (clnt_call(clnt,DELETEW,xdr_wrapstring,argp,xdr_int, &res,TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
return (&res);
}

int *lookupw_1(argp, clnt)
        char **argp; CLIENT *clnt;
{
static int res;
bzero((char *)&res, sizeof(res));
if (clnt_call(clnt,LOOKUPW,xdr_wrapstring,argp,xdr_int, &res, TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        } return (&res);
}
```

primitives utilisées dans les souches serveur :

`pmap_unset()` annule tout lien avec le programme et le numéro de version spécifié.

`svc_udpcreate()` crée et enregistre la référence à une extrêmité de communication, on n'est pas obligé d'indiquer la socket qui va être utilisée pour la communication, dans ce cas on met `RPC_ANYSOCK`.

`svc_register ()` enregistre un programme "d'aiguillage" auprès du daemon portmap.

`svc_sendreply ()` permet d'effectuer une réponse à un client dans un programme d'aiguillage.

Souche Serveur produite par RPCGEN : rdict_svc.c

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "rdict.h"

static void rdictprog_1();

main()
{
        SVCXPRT *transp;

        (void)pmap_unset(RDICTPROG, RDICTVERS);


        transp = svcudp_create(RPC_ANYSOCK);
        if (transp == NULL) {
                (void)fprintf(stderr, "cannot create udp service.\n");
                exit(1);
        }
        if (!svc_register(transp,
                                RDICTPROG,RDICTVERS,rdictprog_1, IPPROTO_UDP)){
                (void)fprintf(stderr,
                                "unable to register (RDICTPROG, RDICTVERS, udp).\n");
                exit(1);
        }

/*      transp = svctcp_create(RPC_ANYSOCK, 0, 0);
        if (transp == NULL) {
                (void)fprintf(stderr, "cannot create tcp service.\n");
                exit(1);
        }
        if (!svc_register(transp, RDICTPROG, RDICTVERS, rdictprog_1, IPPROTO_TCP)) {
                (void)fprintf(stderr, "unable to register (RDICTPROG, RDICTVERS, tcp).\n");
                exit(1);
        }
*/
        svc_run();
        (void)fprintf(stderr, "svc_run returned\n");
        exit(1);
}
```

```
/* squelette de traitement du serveur */
static void rdictprog_1(rqstp, transp)
        struct svc_req *rqstp;
        SVCXPRT *transp;
{
        union {
                char *insertw_1_arg;
                char *deletew_1_arg;
                char *lookupw_1_arg;
        } argument;
        char *result;
        bool_t (*xdr_argument)(), (*xdr_result)();
        char *(*local)();

        switch (rqstp->rq_proc) {
        case NULLPROC:
                (void)svc_sendreply(transp, xdr_void, (char *)NULL);
                return;


        case INITW:
                xdr_argument = xdr_void;
                xdr_result = xdr_int;
                local = (char *(*)()) initw_1;
                break;

        case INSERTW:
                xdr_argument = xdr_wrapstring;
                xdr_result = xdr_int;
                local = (char *(*)()) insertw_1;
                break;

        case DELETEW:
                xdr_argument = xdr_wrapstring;
                xdr_result = xdr_int;
                local = (char *(*)()) deletew_1;
                break;

        case LOOKUPW:
                xdr_argument = xdr_wrapstring;
                xdr_result = xdr_int;
                local = (char *(*)()) lookupw_1;
                break;

        default:
                svcerr_noproc(transp);
                return;
        }
```

```
        bzero((char *)&argument, sizeof(argument));
        if (!svc_getargs(transp, xdr_argument, &argument)) {
                svcerr_decode(transp);
                return;
        }
        result = (*local)(&argument, rqstp); /* appel au code serveur */
        if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
                svcerr_systemerr(transp);
        }
        if (!svc_freeargs(transp, xdr_argument, &argument)) {
                (void)fprintf(stderr, "unable to free arguments\n");
                exit(1);
        }
}
```

Etape 2. Client RPC Sun : rdict.c

```
CLIENT *
clnt_create(host,prognum,versnum,protocol)
   char *host;
   u_long prognum, versum;
   char *protocol;
```

création d'un identificateur de procédure, le protocole est "tcp" ou "udp"

```
/* rdict.c - main, nextin */

#include <rpc/rpc.h>
#include <stdio.h>
#include <ctype.h>
#include "rdict.h"

#defineMAXWORD   50                    /* maximum length of a command or word    */

#define      RMACHINE       "host"/* name of remote machine                */
CLIENT   *handle;                   /* handle for remote procedure */

/*---------------------------------------------------------------------
 * main - insert, delete, or lookup words in a dictionary as specified
 *---------------------------------------------------------------------*/

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

{
        char    word[MAXWORD+1]; /* space to hold word from input line        */
        char    cmd;
        int     wrdlen;                 /* length of input word                 */

     /* initialisation du dialogue pour le RPC */

      handle =
            clnt_create(RMACHINE, RDICTPROG, RDICTVERS, "udp");
      if (handle == 0) {
           printf("Could not contact remote program.\n");
```

```
        exit(1);
    }
```

```
        while (1) {
                wrdlen = nextin(&cmd, word);
                if (wrdlen < 0)
                        exit(0);
                switch (cmd) {
                case 'I':       /* "initialize" */
                        initw();
                        printf("Dictionary initialized to empty.\n");
                        break;
                case 'i':       /* "insert" */
                        insertw(word);
                        printf("%s inserted.\n",word);
                        break;
                case 'd':       /* "delete" */
                        if (deletew(word))
                                printf("%s deleted.\n",word);
                        else
                                printf("%s not found.\n",word);
                        break;
                case 'l':       /* "lookup" */
                        if (lookupw(word))
                                printf("%s was found.\n",word);
                        else
                                printf("%s was not found.\n",word);
                        break;
                case 'q':       /* quit */
                        printf("program quits.\n");
                        exit(0);
                default:/* illegal input */
                        printf("command %c invalid.\n", cmd);
                        break;
                }
        }
}

/*----------------------------------------------------------------------
 * nextin - read a command and (possibly) a word from the next input line
 *----------------------------------------------------------------------*/

int nextin(cmd, word)
char    *cmd, *word;
{
…
}
```

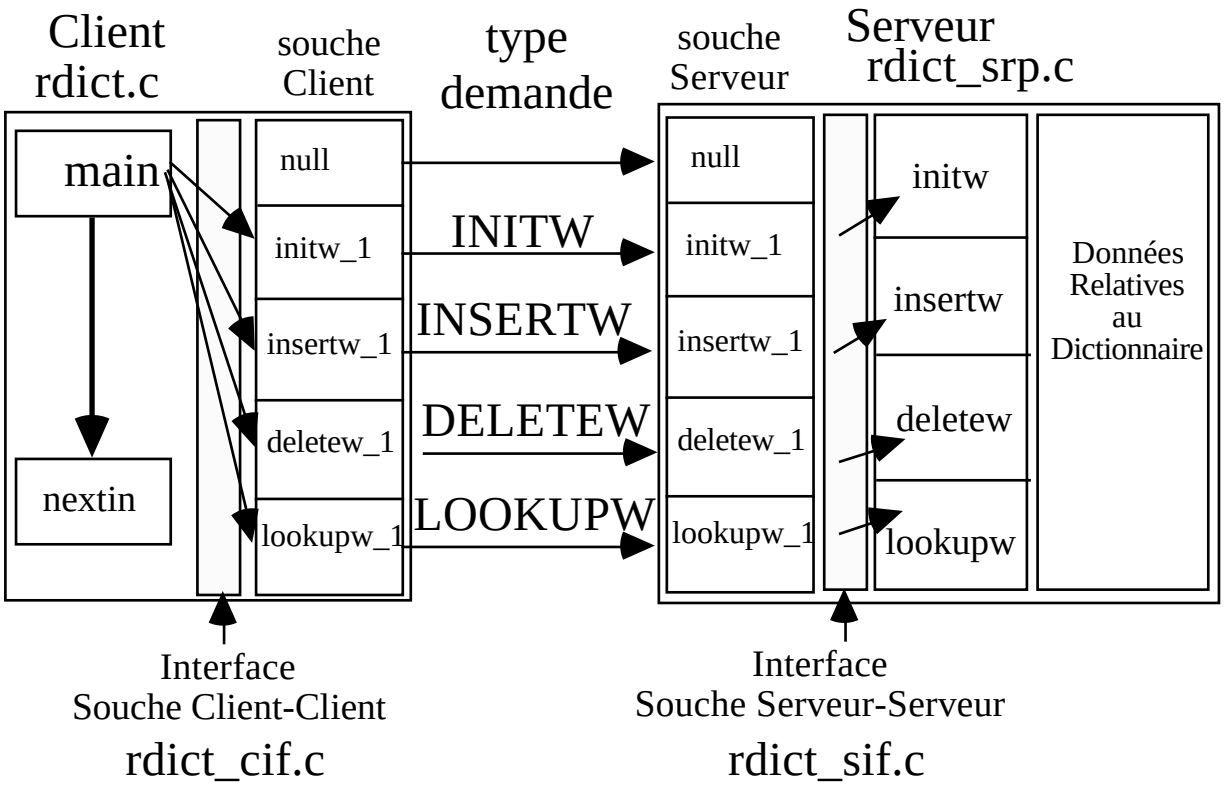
Etape 3. Serveur RPC Sun : rdict_srp.c

Corps des traitements réalisés par le serveur, en terme d'objet, ça serait l'implantation des méthodes.

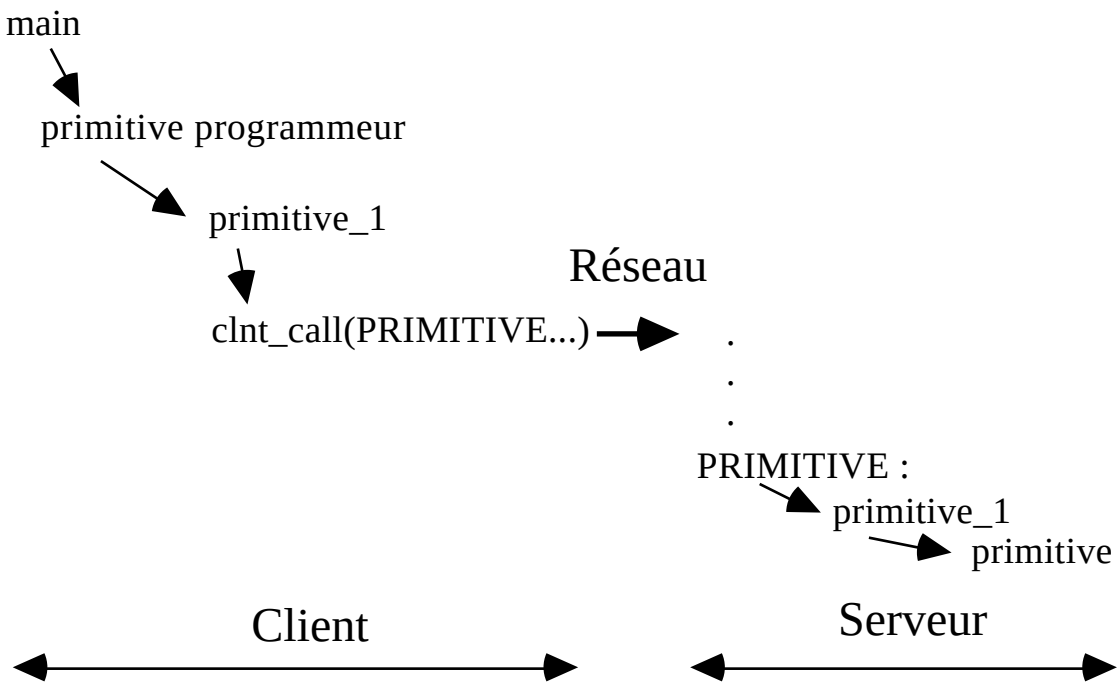
```
/* rdict_srp.c - initw, insertw, deletew, lookupw */
#include <rpc/rpc.h>
#include "rdict.h"

/* Server-side remote procedures and the global data they use */
char    dict[DICTSIZ][MAXWORD+1];       /* storage for a dictionary of words    */
int     nwords = 0;                     /* number of words in the dictionary  */

/*  initw - initialize the dictionary to contain no words at all */
int initw()
{
        nwords = 0;
        return 1;
}

/* insertw - insert  a word in the dictionary */
int insertw(word)
char    *word;
{
        strcpy(dict[nwords], word);
        nwords++;
        return nwords;
}

/* deletew - delete  a word from the dictionary */
int deletew(word)
char    *word;
{
        int     i;
        for (i=0 ; i<nwords ; i++)
                if (strcmp(word, dict[i]) == 0) {
                        nwords--;
                        strcpy(dict[i], dict[nwords]);
                        return 1;
                }
        return 0;
}

/* lookupw - look up a word in the dictionary */
int lookupw(word)
char    *word;
{
        int     i;
        for (i=0 ; i<nwords ; i++)
                if (strcmp(word, dict[i]) == 0)
                        return 1;
        return 0;
```

```
}
```

Interface avec les souches



Cheminement de l'appel du client jusqu'à l'exécution sur le serveur :



Interface Client-Souche Client

```
/* rdict_cif.c - initw, insertw, deletew, lookupw */
#include <rpc/rpc.h>
#include <stdio.h>
#include "rdict.h"

/* Client-side stub interface routines written by hand */
extern      CLIENT    *handle;    /* handle for remote procedure */

/*----------------------------------------------------------------------
 * initw - client interface routine that calls initw_1
 *----------------------------------------------------------------------*/
int initw()
{
        char    *arg;   arg = NULL;             /* pointer to argument (no argument)*/
        return *initw_1(arg,handle);
}

/*----------------------------------------------------------------------
 * insertw - client interface routine that calls insertw_1
 *----------------------------------------------------------------------*/
int insertw(word)
char    *word;
{
        char    **arg;  arg = &word;            /* pointer to argument */
        return *insertw_1(arg, handle);
}

/*----------------------------------------------------------------------
 * deletew - client interface routine that calls deletew_1
 *----------------------------------------------------------------------*/
int deletew(word)
char    *word;
{
        char    **arg;  arg = &word;            /* pointer to argument */
        return *deletew_1(arg, handle);
}

/*----------------------------------------------------------------------
 * lookupw - client interface routine that calls lookupw_1
 *----------------------------------------------------------------------*/
int lookupw(word)
char    *word;
{
        char    **arg;  arg = &word;            /* pointer to argument */
        return *lookupw_1(arg, handle);
}
```

Interface Serveur-Souche Serveur

```
/* rdict_sif.c - init_1, insert_1, delete_1, lookup_1 */
#include <rpc/rpc.h>
#include "rdict.h"


/* Server-side stub interface routines written by hand */

static    int retcode;

/*-------------------------------------------------------------------------
 * insertw_1 -  server side interface to remote procedure insertw
 *-------------------------------------------------------------------------*/
int     *insertw_1(w)
char    **w;
{
        retcode = insertw(*w);
        return &retcode;
}

/*-------------------------------------------------------------------------
 * initw_1 -  server side interface to remote procedure initw
 *-------------------------------------------------------------------------*/
int     *initw_1()
{
        retcode = initw();
        return &retcode;
}

/*-------------------------------------------------------------------------
 * deletew_1 -  server side interface to remote procedure deletew
 *-------------------------------------------------------------------------*/
int     *deletew_1(w)
char    **w;
{
        retcode = deletew(*w);
        return &retcode;
}

/*-------------------------------------------------------------------------
 * lookupw_1 -  server side interface to remote procedure lookupw
 *-------------------------------------------------------------------------*/
int     *lookupw_1(w)
char    **w;
{
        retcode = lookupw(*w);
        return &retcode;
}
```

Mise en oeuvre des programmes générés

source dans /local/work/gressier/sourcesinternet/comer/v3.dist/RPC.ex

. Compilation sur le client

```
cc -c rdict_clnt.c
cc -c rdict_cif.c
cc -c rdict.c
cc -c rdict_xdr.c
cc -o rdict rdict_clnt.o rdict_cif.o rdict.o rdict_xdr.o
```

. Compilation sur le serveur

```
cc -c rdict_svc.c
cc -c rdict_sif.c
cc -c rdict_srp.c
cc -c rdict_xdr.c
cc -o rdictd rdict_svc.o rdict_sif.o rdict_srp.o rdict_xdr.o
```

. Lancement du serveur

```
rdictd &
```

. Utilisation par le client

```
rdict I
rdict i Navy
rdict l Navy
```

rdict d Navy