LES R.P.C.

Licence de ce document

Permission est accordée de copier, distribuer et/ou modifier ce document selon les termes de la "Licence de Documentation Libre GNU" (GNU Free Documentation License), version 1.1 ou toute version ultérieure publiée par la Free Software Foundation.
En particulier le verbe "modifier" autorise tout lecteur éventuel à apporter au document ses propres contributions ou à corriger d'éventuelles erreurs et lui permet d'ajouter son propre copyright dans la page "Registre des éditions" mais lui interdit d'enlever les copyrights déjà présents et lui interdit de modifier les termes de cette licence.
Ce document ne peut être cédé, déposé ou distribué d'une autre manière que l'autorise la "Licence de Documentation Libre GNU" et toute tentative de ce type annule automatiquement les droits d'utiliser ce document sous cette licence. Toutefois, des tiers ayant reçu des copies de ce document ont le droit d'utiliser ces copies et continueront à bénéficier de ce droit tant qu'ils respecteront pleinement les conditions de la licence.

La version papier de ce document est la *1.0* et sa version la plus récente est disponible en téléchargement à l'adresse http://www.frederic-lang.fr.fm

Copyright © 2004 Frédéric Lang (frederic-lang@fr.fm)

Registre des éditions

Version	Date	Description des modifications	Auteur des modifications
1.0	20 février 2005	Mise en ligne du document	© Frédéric Lang (frederic-lang@fr.fm)

SOMMAIRE

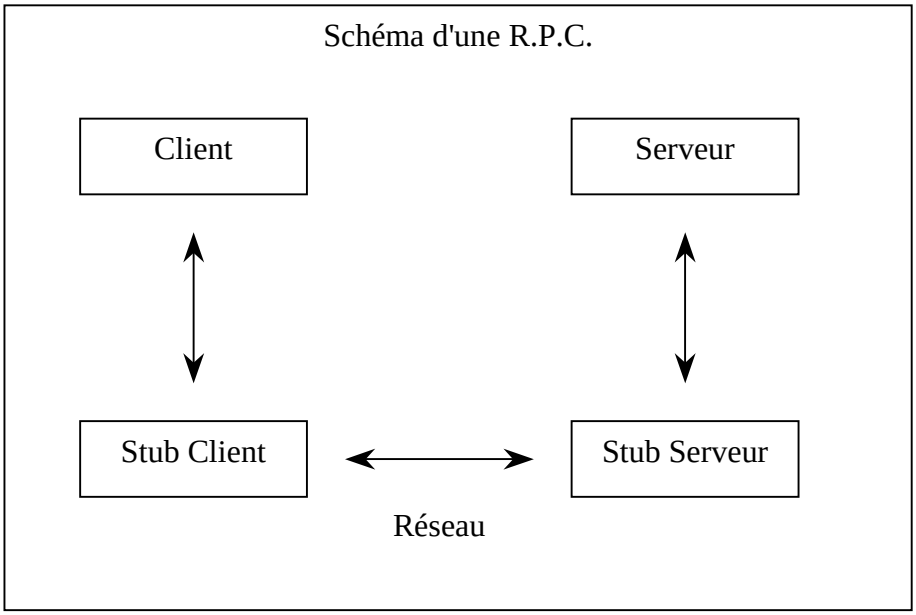
I) GENERALITES

1) Présentation

Les R.P.C. (Remote Procedure Call) sont des outils s'appuyant sur une architecture client-serveur/sockets traditionnelle dans le monde Unix. Leur but est de distribuer des fonctions sur différentes machines appelées "serveur" et de pouvoir les exécuter depuis n'importe quel client. Par exemple, une application devant faire des calculs complexes très rapidement aura tout intérêt à ce que ces calculs complexes soient fait par une machine adaptée à cette tâche; et n'aura plus qu'à appeler la fonction distante et à récupérer le résultat.

2) Principe de fonctionnement

Le système R.P.C. s'efforce de maintenir le plus possible la sémantique habituelle des appels de fonction; autrement dit, tout doit être le plus transparent possible pour le programmeur. Pour que cela ressemble à un appel de fonction classique, il existe, dans le programme client, une fonction locale qui a le même nom que la fonction distante mais qui, en réalité, appelle d'autres fonctions de la bibliothèque R.P.C. qui prennent en charge les connexions réseaux, le passage des paramètres et le retour des résultats. De même, du coté serveur, il suffira d'écrire une fonction comme on en écrit tous les jours, le processus se chargeant d'attendre les connexions clientes et d'appeler votre fonction avec les bons paramètres pour ensuite renvoyer les résultats.
Les fonctions qui prennent en charge les connexions réseaux sont des "stubs". Il faut donc écrire un "stub client" et un "stub serveur" en plus du client et du serveur traditionnels.
Le travail nécessaire à la construction des différents programmes sera automatisé grâce à l'utilisation du programme "**rpcgen**" qui produira du code C. Quelques petites modifications du client et du serveur seront nécessaire pour inclure les besoins du programmeur avant de compiler.



II) EXEMPLE SIMPLE : UNE ADDITION ET UNE MULTIPLICATION

L'exemple qui va être écrit sera de faire faire au serveur une addition et une multiplication de deux entiers positifs, avec remontée d'erreur.

1) Définition de l'interface

Cette interface sera écrite en I.D.L. (Interface Définition Language) du système R.P.C. (proche du C) dans un fichier appelé par exemple "**calcul.x**".

```
/* Déclaration des éléments de travail */
struct s_input {                        /* Structure associée à l'entrée */
        unsigned short nb1;             /* Nombre 1 à opérer */
        unsigned short nb2;             /* Nombre 2 à opérer */
};
typedef struct s_input t_input;         /* Définition nouveau type */

struct s_output {                       /* Structure associé à la sortie */
        unsigned short valeur;          /* Valeur de l'opération */
        int errno;                      /* Cause de l'erreur éventuelle */
};
typedef struct s_output t_output;       /* Définition nouveau type */

/* Définition du programme */
program PROG {                          /* Nom du programme */
        version VERSION_UN {            /* Nom de la version 1 */
                void TEST(void)=0;      /* Fonction de test (obligatoire) */
                t_output ADD(t_input)=1; /* Fonction d'addition */
        }=1;                            /* Numéro de version associé */
        version VERSION_DEUX {          /* Nom de la version 2 */
                void TEST(void)=0;      /* Fonction de test (obligatoire) */
                t_output MULT(t_input)=1; /* Fonction de multiplication */
        }=2;                            /* Numéro de version associé */
}=0x20000001;                           /* Numéro de programme */
```

Les premiers bloc servent à définir la ou les structures d'entrées associées à l'entrée et à la sortie des fonctions crées. A noter que le raccourci "*typedef struct …*" n'est pas utilisable dans l'I.D.L. Ainsi, il a été décidé que les fonctions manipuleraient deux nombres et, afin qu'elles puissent renvoyer plusieurs éléments, ceux-ci sont définis dans une structure *s_output*.

Le second bloc sert à définir le ou les programmes de travail. Chaque programme comporte un nom (ici "**PROG**") et un numéro qui l'identifie de manière unique dans le monde (ici "**0x20000001**"). Les numéros allant de "0x20000001" à "0x3FFFFFFF" sont réservés aux utilisateurs et ne risquent pas d'entrer en conflit avec les programmes déjà existants.
Chaque programme est subdivisé en versions qui comportent elles aussi un nom et un numéro.
Chaque version contient aussi des procédures diverses associées et un numéro. C'est dans la description des procédures que l'on va indiquer le type de données en entrée et en sortie. De plus, chaque version doit contenir une procédure de test ne recevant et ne renvoyant rien, et portant le numéro "0".
Une fois l'interface écrite, il va être traité par l'utilitaire "**rpcgen**" (R.P.C. program generator) grâce à la ligne de commande "*rpcgen –a calcul.x*".
L'option "-a" permet de créer les squelettes des sources "*calcul_server.c*" (source du serveur) et "*calcul_client.c*" (source du client). Indépendamment de ces fichiers, il sera aussi créé "*calcul.h*" (en-tête), "*calcul_clnt.c*" (stub-client), "*calcul_svr.c*" (stub-serveur), "*calcul_xdr.c*" (routines XDR) et "*Makefile.calcul*" (outil de compilation).
Le format X.D.R. (eXternal Data Representation) définit les types utilisés pour l'échange de variables entre le client et le serveur. Il est possible que le client et le serveur ne tournent pas sur la même plate-forme.; il est donc indispensable de parler une langue commune. Ainsi, ce format définit précisément un codage pour les entiers, les flottants, etc.
Ce source peut être compilé et lié avec le serveur et le client.

2) Fichiers générés

a) Fichiers communs

Fichier "*Makefile.calcul*" (à ne pas modifier)

```
# This is a template Makefile generated by rpcgen

# Parameters

CLIENT = calcul_client
SERVER = calcul_server

SOURCES_CLNT.c =
SOURCES_CLNT.h =
SOURCES_SVC.c =
SOURCES_SVC.h =
SOURCES.x = calcul.x

TARGETS_SVC.c = calcul_svc.c calcul_server.c calcul_xdr.c
TARGETS_CLNT.c = calcul_clnt.c calcul_client.c calcul_xdr.c
TARGETS = calcul.h calcul_xdr.c calcul_clnt.c calcul_svc.c calcul_client.c calcul_server.c

OBJECTS_CLNT = $(SOURCES_CLNT.c:%.c=%.o) (TARGETS_CLNT.c:%.c=%.o)
OBJECTS_SVC = $(SOURCES_SVC.c:%.c=%.o) $(TARGETS_SVC.c:%.c=%.o)
# Compiler flags

CFLAGS += -g
LDLIBS += -lnsl
RPCGENFLAGS =

# Targets

all : $(CLIENT) $(SERVER)

$(TARGETS) : $(SOURCES.x)
	rpcgen $(RPCGENFLAGS) $(SOURCES.x)

$(OBJECTS_CLNT) : $(SOURCES_CLNT.c) $(SOURCES_CLNT.h) $(TARGETS_CLNT.c)

$(OBJECTS_SVC) : $(SOURCES_SVC.c) $(SOURCES_SVC.h) $(TARGETS_SVC.c)

$(CLIENT) : $(OBJECTS_CLNT)
	$(LINK.c) -o $(CLIENT) $(OBJECTS_CLNT) $(LDLIBS)

$(SERVER) : $(OBJECTS_SVC)
	$(LINK.c) -o $(SERVER) $(OBJECTS_SVC) $(LDLIBS)

clean:
	 $(RM) core $(TARGETS) $(OBJECTS_CLNT) $(OBJECTS_SVC) $(CLIENT)
$(SERVER)
```

Fichier "*calcul.h*" (à ne pas modifier)

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

#ifndef _CALCUL_H_RPCGEN
```

```
#define _CALCUL_H_RPCGEN
#include <rpc/rpc.h>


#ifdef __cplusplus
extern "C" {
#endif


struct s_input {
        u_short nb1;
        u_short nb2;
};
typedef struct s_input s_input;

typedef s_input t_input;

struct s_output {
        u_short valeur;
        int errno;
};
typedef struct s_output s_output;

typedef s_output t_output;

#define PROG 0x20000001
#define VERSION_UN 1

#if defined(__STDC__) || defined(__cplusplus)
#define TEST 0
extern void * test_1(void *, CLIENT *);
extern void * test_1_svc(void *, struct svc_req *);
#define ADD 1
extern t_output * add_1(t_input *, CLIENT *);
extern t_output * add_1_svc(t_input *, struct svc_req *);
extern int prog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

#else /* K&R C */
#define TEST 0
extern void * test_1();
extern void * test_1_svc();
#define ADD 1
extern t_output * add_1();
extern t_output * add_1_svc();
extern int prog_1_freeresult ();
#endif /* K&R C */
#define VERSION_DEUX 2

#if defined(__STDC__) || defined(__cplusplus)
extern void * test_2(void *, CLIENT *);
extern void * test_2_svc(void *, struct svc_req *);
#define MULT 1
extern t_output * mult_2(t_input *, CLIENT *);
extern t_output * mult_2_svc(t_input *, struct svc_req *);
extern int prog_2_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

#else /* K&R C */
extern void * test_2();
extern void * test_2_svc();
#define MULT 1
extern t_output * mult_2();
```

```
extern t_output * mult_2_svc();
extern int prog_2_freeresult ();
#endif /* K&R C */

/* the xdr functions */

#if defined(__STDC__) || defined(__cplusplus)
extern bool_t xdr_s_input (XDR *, s_input*);
extern bool_t xdr_t_input (XDR *, t_input*);
extern bool_t xdr_s_output (XDR *, s_output*);
extern bool_t xdr_t_output (XDR *, t_output*);

#else /* K&R C */
extern bool_t xdr_s_input ();
extern bool_t xdr_t_input ();
extern bool_t xdr_s_output ();
extern bool_t xdr_t_output ();

#endif /* K&R C */

#ifdef __cplusplus
}
#endif

#endif /* !_CALCUL_H_RPCGEN */
```

Fichier "*calcul_xdr.c*" (à ne pas modifier)

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

#include "calcul.h"

bool_t xdr_s_input (XDR *xdrs, s_input *objp)
{
        register int32_t *buf;

         if (!xdr_u_short (xdrs, &objp->nb1))
                 return FALSE;
         if (!xdr_u_short (xdrs, &objp->nb2))
                 return FALSE;
        return TRUE;
}

bool_t xdr_t_input (XDR *xdrs, t_input *objp)
{
        register int32_t *buf;

         if (!xdr_s_input (xdrs, objp))
                 return FALSE;
        return TRUE;
}

bool_t xdr_s_output (XDR *xdrs, s_output *objp)
{
        register int32_t *buf;

         if (!xdr_u_short (xdrs, &objp->valeur))
                 return FALSE;
         if (!xdr_int (xdrs, &objp->errno))
```

```
        return FALSE;
}

bool_t xdr_t_output (XDR *xdrs, t_output *objp)
{
        register int32_t *buf;

         if (!xdr_s_output (xdrs, objp))
                 return FALSE;
        return TRUE;
}
```

b) Fichiers serveurs

Fichier "*calcul_svr.c*" (à ne pas modifier)

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

#include "calcul.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifndef SIG_PF
#define SIG_PF void(*)(int)
#endif

static void prog_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
	union {
		t_input add_1_arg;
	} argument;
	char *result;
	xdrproc_t _xdr_argument, _xdr_result;
	char *(*local)(char *, struct svc_req *);

	switch (rqstp->rq_proc) {
	case TEST:
		_xdr_argument = (xdrproc_t) xdr_void;
		_xdr_result = (xdrproc_t) xdr_void;
		local = (char *(*)(char *, struct svc_req *)) test_1_svc;
		break;

	case ADD:
		_xdr_argument = (xdrproc_t) xdr_t_input;
		_xdr_result = (xdrproc_t) xdr_t_output;
		local = (char *(*)(char *, struct svc_req *)) add_1_svc;
		break;

	default:
		svcerr_noproc (transp);
		return;
	}
	memset ((char *)&argument, 0, sizeof (argument));
	if (!svc_getargs (transp, _xdr_argument, (caddr_t) &argument)) {
		svcerr_decode (transp);
		return;
	}
	result = (*local)((char *)&argument, rqstp);
	if (result != NULL && !svc_sendreply(transp, _xdr_result, result)) {
		svcerr_systemerr (transp);
	}
	if (!svc_freeargs (transp, _xdr_argument, (caddr_t) &argument)) {
		fprintf (stderr, "unable to free arguments");
		exit (1);
	}
	return;
}
```

```
static void prog_2(struct svc_req *rqstp, register SVCXPRT *transp)
{
        union {
                t_input mult_2_arg;
        } argument;
        char *result;
        xdrproc_t _xdr_argument, _xdr_result;
        char *(*local)(char *, struct svc_req *);

        switch (rqstp->rq_proc) {
        case TEST:
                _xdr_argument = (xdrproc_t) xdr_void;
                _xdr_result = (xdrproc_t) xdr_void;
                local = (char *(*)(char *, struct svc_req *)) test_2_svc;
                break;

        case MULT:
                _xdr_argument = (xdrproc_t) xdr_t_input;
                _xdr_result = (xdrproc_t) xdr_t_output;
                local = (char *(*)(char *, struct svc_req *)) mult_2_svc;
                break;

        default:
                svcerr_noproc (transp);
                return;
        }
        memset ((char *)&argument, 0, sizeof (argument));
        if (!svc_getargs (transp, _xdr_argument, (caddr_t) &argument)) {
                svcerr_decode (transp);
                return;
        }
        result = (*local)((char *)&argument, rqstp);
        if (result != NULL && !svc_sendreply(transp, _xdr_result, result)) {
                svcerr_systemerr (transp);
        }
        if (!svc_freeargs (transp, _xdr_argument, (caddr_t) &argument)) {
                fprintf (stderr, "unable to free arguments");
                exit (1);
        }
        return;
}

int main (int argc, char **argv)
{
        register SVCXPRT *transp;

        pmap_unset (PROG, VERSION_UN);
        pmap_unset (PROG, VERSION_DEUX);

        transp = svcudp_create(RPC_ANYSOCK);
        if (transp == NULL) {
                fprintf (stderr, "cannot create udp service.");
                exit(1);
        }
        if (!svc_register(transp, PROG, VERSION_UN, prog_1, IPPROTO_UDP)) {
                fprintf (stderr, "unable to register (PROG, VERSION_UN, udp).");
                exit(1);
        }
        if (!svc_register(transp, PROG, VERSION_DEUX, prog_2, IPPROTO_UDP)) {
                fprintf (stderr, "unable to register (PROG, VERSION_DEUX, udp).");
                exit(1);
```

```
        }
        transp = svctcp_create(RPC_ANYSOCK, 0, 0);
        if (transp == NULL) {
                fprintf (stderr, "cannot create tcp service.");
                exit(1);
        }
        if (!svc_register(transp, PROG, VERSION_UN, prog_1, IPPROTO_TCP)) {
                fprintf (stderr, "unable to register (PROG, VERSION_UN, tcp).");
                exit(1);
        }
        if (!svc_register(transp, PROG, VERSION_DEUX, prog_2, IPPROTO_TCP)) {
                fprintf (stderr, "unable to register (PROG, VERSION_DEUX, tcp).");
                exit(1);
        }

        svc_run ();
        fprintf (stderr, "svc_run returned");
        exit (1);
        /* NOTREACHED */
}
```

Fichier "*calcul_server.c*" (sera modifié ultèrieurement)

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "calcul.h"

void *test_1_svc(void *argp, struct svc_req *rqstp)
{
        static char * result;

        /*
         * insert server code here
         */

        return (void *) &result;
}

t_output *add_1_svc(t_input *argp, struct svc_req *rqstp)
{
        static t_output result;

        /*
         * insert server code here
         */

        return &result;
}

void *test_2_svc(void *argp, struct svc_req *rqstp)
{
        static char * result;

        /*
         * insert server code here
         */
```

```
}       return (void *) &result;

t_output *mult_2_svc(t_input *argp, struct svc_req *rqstp)
{
        static t_output result;

        /*
         * insert server code here
         */

        return &result;
}
```

c) Fichiers clients

Fichier "*calcul_clnt.c*" (à ne pas modifier)

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

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

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

void *test_1(void *argp, CLIENT *clnt)
{
        static char clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, TEST,
                (xdrproc_t) xdr_void, (caddr_t) argp,
                (xdrproc_t) xdr_void, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return ((void *)&clnt_res);
}

t_output *add_1(t_input *argp, CLIENT *clnt)
{
        static t_output clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, ADD,
                (xdrproc_t) xdr_t_input, (caddr_t) argp,
                (xdrproc_t) xdr_t_output, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}

void *test_2(void *argp, CLIENT *clnt)
{
        static char clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, TEST,
                (xdrproc_t) xdr_void, (caddr_t) argp,
                (xdrproc_t) xdr_void, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return ((void *)&clnt_res);
}

t_output *mult_2(t_input *argp, CLIENT *clnt)
{
        static t_output clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, MULT,
```

```
                (xdrproc_t) xdr_t_input, (caddr_t) argp,
                (xdrproc_t) xdr_t_output, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```

Fichier "*calcul_client.c*" (sera modifié ultèrieurement)

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "calcul.h"


void prog_1(char *host)
{
        CLIENT *clnt;
        void *result_1;
        char *test_1_arg;
        t_output *result_2;
        t_input add_1_arg;

#ifndef        DEBUG
        clnt = clnt_create (host, PROG, VERSION_UN, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif/* DEBUG */

        result_1 = test_1((void*)&test_1_arg, clnt);
        if (result_1 == (void *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        result_2 = add_1(&add_1_arg, clnt);
        if (result_2 == (t_output *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef        DEBUG
        clnt_destroy (clnt);
#endif /* DEBUG */
}


void prog_2(char *host)
{
        CLIENT *clnt;
        void *result_1;
        char *test_2_arg;
        t_output *result_2;
        t_input mult_2_arg;

#ifndef        DEBUG
        clnt = clnt_create (host, PROG, VERSION_DEUX, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
```

```
}
#endif /* DEBUG */

        result_1 = test_2((void*)&test_2_arg, clnt);
        if (result_1 == (void *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        result_2 = mult_2(&mult_2_arg, clnt);
        if (result_2 == (t_output *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef         DEBUG
        clnt_destroy (clnt);
#endif /* DEBUG */
}


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

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        prog_1(host);
        prog_2(host);
exit (0);
}
```

3) Modifications à apporter

a) Serveur

Les fonctions "*test_1_svc*" et "*test_2_svc*" servent à tester le bon fonctionnement du serveur. Un simple "*printf()*" dans leur code suffira.

```
void *test_1_svc(void *argp, struct svc_req *rqstp)
{
        printf("%s – Test version1\n", __FILE__);
}


void *test_2_svc(void *argp, struct svc_req *rqstp)
{
        printf("%s – Test version 2\n", __FILE__);
}
```

Les fonctions "*add_1_svc*" et "*mult_2_svc*" seront les fonctions effectuant respectivement l'addition et la multiplication. Chacune recevra en paramètre un pointeur de type *"t_input*" nommé "*argp*" et renverra un pointeur de type "*t_output*" nommé "*result*". C'est au programmeur de coder chaque calcul et de remplir la variable retournée.

```
t_output *add_1_svc(t_input *argp, struct svc_req *rqstp)
{
        /* Variable statique pour garantir son existence au retour de fonction */
        static t_output result;

        /* Addition */
        result.valeur=argp->nb1 + argp->nb2;

        /* Si résultat < max(nb1,nb2) */
        if (result.valeur < (argp->nb1 > argp->nb2 ?argp->nb1 :argp->nb2))
        {
                /* Overflow */
                result.errno=1;
        }
        else
        {
                /* Pas d'erreur */
                result.errno=0;
        }

        /* Renvoi du résultat */
        return(&result);
}

t_output *mult_2_svc(t_input *argp, struct svc_req *rqstp)
{
        /* Variable statique pour garantir son existence au retour de fonction */
        static t_output result;

        /* Multiplication */
        result.valeur=argp->nb1 * argp->nb2;

        /* Si résultat < max(nb1,nb2) */
        if (result.valeur < (argp->nb1 > argp->nb2 ?argp->nb1 :argp->nb2))
        {
                /* Overflow */
                result.errno=1;
```

```
        }
        else
        {
                /* Pas d'erreur */
                result.errno=0;
        }

        /* Renvoi du résultat */
        return(&result);
}
```

Remarque: La fonction renvoyant un pointeur sur une variable **locale** (*result*), celle-ci doit impérativement être déclarée en "*static*" afin de garantir son existence lorsque la fonction sera dépilée.

b) Client

Les fonctions "*prog_1*" et "*prog_2*" seront les fonctions appelant l'addition et la multiplication. Elles n'ont pas été prévues pour recevoir de paramètres en dehors du nom du serveur; ni de renvoyer de valeur. Le programmeur doit donc les modifier pour permettre aux nombres à opérer d'arriver jusqu'à la fonction et au calcul de ressortir.

Fonctions originales

```
void prog_1(char *host)
{
        CLIENT *clnt;
        void *result_1;
        char *test_1_arg;
        t_output *result_2;
        t_input add_1_arg;

#ifndef        DEBUG
        clnt = clnt_create (host, PROG, VERSION_UN, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif/* DEBUG */

        result_1 = test_1((void*)&test_1_arg, clnt);
        if (result_1 == (void *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        result_2 = add_1(&add_1_arg, clnt);
        if (result_2 == (t_output *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef        DEBUG
        clnt_destroy (clnt);
#endif /* DEBUG */
}


void prog_2(char *host)
{
        CLIENT *clnt;
        void *result_1;
        char *test_2_arg;
        t_output *result_2;
        t_input mult_2_arg;

#ifndef        DEBUG
        clnt = clnt_create (host, PROG, VERSION_DEUX, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif/* DEBUG */

        result_1 = test_2((void*)&test_2_arg, clnt);
        if (result_1 == (void *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        result_2 = mult_2(&mult_2_arg, clnt);
        if (result_2 == (t_output *) NULL) {
                clnt_perror (clnt, "call failed");
```

```
}
#ifndef         DEBUG
        clnt_destroy (clnt);
#endif /* DEBUG */
}
```

Fonctions modifiées

```
t_output *prog_1(char *host, t_input *arg)
{
        CLIENT *clnt;
        t_output *result;

#ifndef         DEBUG
        clnt = clnt_create (host, PROG, VERSION_UN, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif/* DEBUG */

        /* Appel de l'addition */
        result = add_1(arg, clnt);
        if (result == (t_output *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef         DEBUG
        clnt_destroy (clnt);
#endif /* DEBUG */

        return(result);
}


t_output *prog_2(char *host, t_input *arg)
{
        CLIENT *clnt;
        t_output *result;

#ifndef         DEBUG
        clnt = clnt_create (host, PROG, VERSION_DEUX, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif/* DEBUG */

        /* Appel de la multiplication */
        result = mult_2(arg, clnt);
        if (result == (t_output *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef         DEBUG
        clnt_destroy (clnt);
#endif /* DEBUG */

        return(result);
}
```

Remarque: L'appel aux fonctions "*test_1*" et "*test_2*", génératrices d'affichage parasite et de ralentissement, a été supprimé.

La fonction "*main()*" du client doit aussi être modifiée. Il est nécessaire de générer les valeurs à opérer avant d'appeler les fonction s"*prog_1*" ou "*prog_2*".

Fonction originale

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

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        prog_1(host);
        prog_2(host);
exit (0);
}
```

Fonction modifiée

```
#include <limits.h>                         /* Limites système pré-définies */
int main (int argc, char *argv[])
{
        char *host;
        t_input arg;
        t_output *result;

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];

        /* Remplissage des valeurs */
        arg.nb1=USHRT_MAX – 15;
        arg.nb2=10;

        /* Addition et vérification */
        result=prog_1(host, &arg);
        if (result->errno != 0)
        {
                /* Erreur de calcul */
                printf("Overflow sur l'addition de %u et %u\n", arg.nb1, arg.nb2);
        }
        else
        {
                /* Affichage du résultat */
                printf("Le résultat de l'addition de %u et %u est %u\n", arg.nb1, arg.nb2, result-
>valeur);
        }

        /* Multiplication et vérification */
        result=prog_2(host, &arg);
        if (result->errno != 0)
        {
                /* Erreur de calcul */
                printf("Overflow sur la multiplication de %u et %u\n", arg.nb1, arg.nb2);
        }
        else
        {
```

```
                /* Affichage du résultat */
                printf("Le résultat de la multiplication de %u et %u est %u\n", arg.nb1, arg.nb2,
result->valeur);
        }

        exit (0);
}
```

4) Exécution

a) Compilation

La compilation totale se fait objet par objet

- ✓ gcc –c calcul_xdr.c
- ✓ gcc –c calcul_svr.c
- ✓ gcc –c calcul_clnt.c
- ✓ gcc –c calcul_client.c
- ✓ gcc –c calcul_server.c

Puis, on génère les exécutables

- ✓ gcc calcul_svr.o calcul_xdr.o calcul_server.o –lnsl –o calcul_server
- ✓ gcc calcul_clnt.o calcul_xdr.o calcul_client.o –lnsl –o calcul_client

On peut aussi utiliser l'outil "*make*" et le fichier de configuration *"Makefile.calcul*"

- ✓ make -f Makefile.calcul calcul_server calcul_client

On peut aussi utiliser, dans l'outil "*make*" et le fichier de configuration *"Makefile.calcul*", la pseudo cible all

- ✓ make -f Makefile.calcul all

a) Lancement du serveur et vérification

Le lancement du serveur se fait avec le programme "*calcul_server*" nouvellement généré par la commande

- ✓ ./calcul_server &

L'utilisation de l'arrière-plan est facultatif. La commande "*rpcinfo –p*" permet de voir que le serveur est effectivement lancé.

b) Vérification du client et lancement

La vérification du client peut déjà se faire avec la commande "*rpcinfo*" en utilisant l'option "*-u*". Cette commande lance les fonctions de tests programmées pour répondre avec un "*printf*".

✓ rpcinfo –u localhost 536870913 ((*localhost* représente le nom de la machine sur laquelle tourne le serveur, le nombre "*536870913*" correspond au nombre hexadécimal "*0x20000001*").

Le lancement du client se fait par la commande

✓ ./calcul_client localhost

INDEX