Dossier de programmation RPC

**TDM3 - Introduction aux RPC**

1 - Nous avons tout d’abord réaliser un programme en langage C (exo1.c), permettant d’écrire dans le fichier poubelle /dev/null. (voir Annexe 3.1.1)

Après avoir testé ce programme, nous avons pu mettre en oeuvre les RPC.

2 - Afin d’écrire un message dans le fichier poubelle d’une autre machine en utilisant les RPC, nous avons réalisé le fichier exo1.x qui définit les caractéristiques du service distant (nom du programme, version, fonction(s) distante(s)) (voir Annexe 3.2.1)

Lorsque l’on passe ce fichier au compilateur rpcgen (avec la commande rpgcgen exo1.x), celui-ci génère les fichiers suivants :

* exo1.h, qui contient les structures, variables et en-têtes
* exo1\_clnt.c, talon client
* exo1\_svc.c, talon serveur
* exo1\_xdr.c

Il faut ensuite réaliser les fichiers rexo1.c (voir Annexe 3.2.2) et exo1\_proc.c (voir Annexe 3.2.3) qui sont respectivement les programmes client et serveur. En effet, le fichier rexo1.c contient la procédure cliente et établit un lien entre le client et le serveur; le fichier exo1\_proc.c contient le code des fonctions distantes qui sera exécuté par le serveur. La procédure distante prend en paramètres un pointeur sur les paramètres qu’elle aurait eu en local (de même pour le retour).

Pour générer les exécutables serveur et client, il faut taper sur le terminal :

* gcc exo1\_svc.c exo1\_proc.c exo1\_xdr.c -o exo1\_svc (exécutable serveur)
* gcc rexo1.c exo1\_clnt.c exo1\_xdr.c -o rexo1 (exécutable client)

Difficultés :

Nous n’avons pas eu de problème particulier sur ce TP, si ce n’est la compréhension générale des RPC.

**TDM4 : Utilisation avancée des RPC**

1 - Nous avons tout d’abord écrit un programme C qui fait la somme des nombres entiers appartenant à un intervalle [x,y] donné par l'utilisateur. (voir Annexe 4.1.1)

2 - Afin de calculer la somme par une autre machine, nous avons réaliser le fichier somme.x (voir Annexe 4.2.1) qui décrit qui définit les caractéristiques des procédures déportées. Pour transférer les deux paramètres (bornes de l’intervalle), nous avons ajouter une structure contenant les deux paramètres (borne\_inf et borne\_sup).

Lorsque nous avons passé le fichier somme.x au compilateur rpcgen (avec la commande “rpcgen somme.x”), nous avons obtenu les fichiers suivants :

* somme.h
* somme\_clnt.c
* somme\_svc.c
* somme\_xdr.c

Il a ensuite fallu crée les deux programmes rsomme.c (voir Annexe 4.2.2) et somme\_proc.c (voir Annexe 4.2.3) qui sont respectivement les programmes client et serveur. Le fichier rsomme.c contient la procédure cliente qui créé l’objet client, d’établir le lien entre client et serveur et de faire appel à la procédure distante. Le reste du programme client gère les erreurs d’appel et de traitements de la fonction déportée.

Le fichier somme\_proc.c contient le code de la fonction calculsomme décrite dans le fichier somme.x, qui sera exécuté par le serveur. Il ne faut pas oublier d’importer le fichier somme.h dans les deux fichiers, qui contient les en têtes et structures. La procédure prend en paramètre un pointeur vers les bornes.

Nous avons ensuite généré les fichiers exécutables client et serveur avec les commandes suivantes sur le terminal :

* gcc rsomme.c somme\_clnt.c somme\_xdr.c -o rsomme (génère l’exécutable appelant/client)
* gcc somme\_svc.c somme\_proc.c somme\_xdr.c -o somme\_svc (génère l’exécutable appelé/serveur)

Pour tester nos programmes, nous avons lancer les commandes suivantes sur deux fenêtres terminal différentes:

* ./somme\_svc (lancement du programme serveur avant le lancement du programme client, sinon cela ne fonctionnera pas)
* ./rsomme localhost 3 5 (lancement du programme client qui renvoi 12 avec les valeurs données en exemple)

Difficultés :

Nous avons rencontrés plus de difficultés lors de ce TP que pour le précédent du fait qu’il fallait envoyer plusieurs paramètres au serveur pour exécuter la fonction. Nous avons également eu un problème lors de l’affichage du résultat de la somme car la variable result n’était pas remise à zéro, le résultat du test précédent était donc conservé. Nous obtenions ainsi des résultats qui augmentaient indéfiniment pour des tests avec des valeurs identiques. Pour corriger ce problème, nous avons ajouter la ligne result = 0 avant la boucle for, ce qui permet de réinitialiser la variable result à chaque exécution.

**TDM5-6 : Étude de cas**

**ANNEXES**

**Annexe 3.1.1 : exo1.c**

FILE \* f;

f = fopen("/dev/null", "w");

if (f == NULL) {

result = 0;

}

// On ecrit le msg passé en param dans le fichier f

fprintf (f, "%s\n", msg);

fclose(f);

return(1);

}

main (int argc, char \*\*argv) {

char \* message;

if (argc!=2) {

perror(“Le nombre de paramètres est invalide”);

exit(1);

}

message = argv[1];

if ( ! printmessage(message)) {

perror(“Impossible d’imprimer le message”);

exit(1);

}

printf("Message imprimer \n");

exit(0);

}

**Annexe 3.2.1 : exo1.x**

program MESSAGEPROG { /\* Nom du prgm \*/

version MESSAGEVERS { /\* version \*/

int PRINTMESSAGE(string)=1; /\* fonction distantes \*/

}=1;

}=0x20000001;

**Annexe 3.2.2 : rexo1.c**

# include <stdio.h>

# include <rpc/rpc.h>

# include “exo1.h”

main (int argc, char \*\*argv) {

CLIENT \*cl;

int \*result;

char \*message;

if (argc != 4) {

perror(“Le nombre de paramètres est invalide”);

exit(1);

}

message = argv[3];

cl = clnt\_create(argv[1], MESSAGEPROG, MESSAGEVERS, "tcp");

if (cl == NULL) {

// impossible de se connecter au serveur, affichage d'un msg erreur et fin

clnt\_pcreateerror(argv[1]);

exit(1);

}

// Appel de la procedure distante "printmessage"

result = printmessage\_1(&message, cl);

if (result == NULL) {

// Erreur lors de l appel du service. Affichage d'un msg erreur puis fin

clnt\_perror(cl, argv[1]);

exit(1);

}

if (\*result == 0) {

perror(“Impossible d’imprimer le message”);

exit(1);

}

/\* le message a ete imprime sur le /dev/null du serveur \*/

printf("Le message a bien été délivré \n");

exit(0);

}

**Annexe 3.2.3 : exo1\_proc.c**

# include <stdio.h>

# include <rpc/rpc.h>

# include "exo1.h" // généré auto par rpcgen, commun au serveur et au client

// centralise les structures, variables, etc ("Pour que svc et clnt soient en phase")

int \*printmessage\_1\_svc(char \*\*msg, struct svc\_req \*rqstp) {

static int result;

FILE\* f;

f = fopen("/dev/null", "w");

if (f == NULL) {

result = 0;

return(&result);

}

// On ecrit le msd passé en param dans f

fprintf(f, "%s\n", msg);

fclose(f);

result = 1;

return(&result);

}

**Annexe 4.1.1 : somme.c**

# include <stdio.h>

# include <stdlib.h>

int calculSomme(int argc, char \*argv[]) {

int result = 0;

int i;

// Conversion des nb entrés par l'util de ascii à integer

//+ transfert des params dans les variables

int x = atoi(argv[1]);

int y = atoi(argv[2]);

for(i=x; i<=y; i++) {

// EX : [3,5] : 0+3, 3+4, 7+5 => 12

result = result + i;

}

printf("résultat de la somme : %d\n", result);

}

**Annexe 4.2.1 : somme.x**

/\* structure contenant les bornes de l’intervalle \*/

struct param{int borne\_inf;

int borne\_sup; };

program SOMME\_PROG {

version SOMME\_VERS {

int CALCULSOMME(param)=1;

}=1;

}=0x20000001;

**Annexe 4.2.2 : rsomme.c**

# include <rpc/rpc.h>

# include "somme.h"

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

CLIENT \*cl;

int \*result;

// definir un obj de type param

param borne;

int x,y;

if (argc != 4){

perror("Nb de param incorrect");

exit(1);

}

// On recup les arg passes en parametres

// atoi means ascii to integer

x = atoi(argv[2]);

y = atoi(argv[3]);

// On affecte les variables x et y aux bornes de l’intervalle

borne.borne\_inf = x;

borne.borne\_sup = y;

// Crea d un obj client

cl = clnt\_create(argv[1], SOMME\_PROG, SOMME\_VERS, "tcp");

if(cl == NULL) {

// impossible de se connecter au serveur, affichage d'un msg erreur et fin

clnt\_pcreateerror(argv[1]);

exit(1);

}

result = calculsomme\_1(&borne,cl); // calculsomme\_1 ds clnt

// calculsomme\_1\_svc dans somme\_proc

if(result == NULL) {

// Erreur lors de l appel du service. Affichage d'un msg erreur puis fin

clnt\_perror(cl, argv[1]);

exit(1);

}

if(\*result == 0) {

perror("Impossible d'obtenir le rslt de la proc distante");

exit(1);

}

// Affichage du résultat de la fonction distante

printf("Le résultat de la somme de l\'interval [%d, %d] est : %d\n", x, y, \*result);

exit(0);

}

**Annexe 4.2.3 : somme\_proc.c**

# include <rpc/rpc.h>

# include "somme.h"

int \* calculsomme\_1\_svc(param \*borne, struct svc\_req \* rqstp) {

static int result;

// traitement

int i;

// Conversion des nb entrés par l'util de ascii à integer

// + transfert des params dans les variables

int x,y;

x = borne->borne\_inf;

y = borne->borne\_sup;

result = 0;

/\* on met result=0 sinon quand on relancera le prgm il reprendra le resultat de l'exec precedente \*/

for(i=x; i<=y; i++) {

// EX : [3,5] : 0+3, 3+4, 7+5 => 12

result = result + i;

}

return(&result);

}