SRIP4 28/03/2014

FLN1147A FLEYTOUX YOANN

TP3-4 RPC

Introduction :

L'objectif de ce TP est de mettre ne œuvre une application client-serveur en utilisant les technologies RPC vues en cours.
Ainsi à travers le tp3 on cherche à développer un client proposant trois fonctionnalités à distance étant détaillé ainsi par le sujet :

-ls(nom_repertoire) : recupere la liste des noms des fichiers (et repertoires) du repertoire <nom_repertoire> et l'envoie au client

-read(nom_fichier) : lit un fichier par blocs de 1024 octets

-write(nom_fichier, ecraser, donnees) : cree un fichier (si ecraser = 1) ou ajoute a la fin d'un fichier les donnees envoyees en parametres (les donnees sont organisees en liste simplement chainee comme au 1.2)

L'énoncé du sujet amenait à la solution, donnant des indications sur les structures des données à employer, ainsi que des méthodes et des librairies.

Représentation des données manipulées par les programmes client et serveur

Une fois les structures de données conçues, elles sont mis en place dans le fichier tp3.x qui en utilisant la commande : « *rpcgen tp3.x* » qui génèrera les talons serveurs et clients**,** *tp3_svc.c* et *tp3_clnt*, le header contenant les différentes fonctions et types de données tp3.h, et XDR qui se chargera de transmettre les différents types de données entre le serveur et le client *tp3_xdr.c*. Ici on utilise donc une liste chainée « liste » dont le pointeur de tête sera renvoyé par les fonctions *ls()* et *read()*, la fonction *writte()* nécessitant 3 arguments en entrée, on passe par une structure « *arg_writte* » pour contourner la limitation qu'impose RPC.

Bilan des difficultés rencontrées

Etant seul, j'ai rencontré des difficultés sur la mise en œuvre du code , restant souvent bloqué sur des problèmes car trop concentré sur la mauvaise partis du code alors que la solution au problème est ailleurs. Je n'ai ainsi pas réussi à résoudre un problème sur le read() qui fonctionnait à un moment mais après modification ne marchait plus du tout et je n'ai pu retrouver une version qui fonctionnait correctement, cependant le reste des fonctionnalités devrais fonctionnés (malheureusement le writte() se repose sur le read() pour récupérer les données à écrire).

Les technologies RPC sont assez faciles à prendre en main est la conception n'a pas trop posé de problèmes. J'ai cependant eu des problèmes pour pouvoir travailler chez moi, n'arrivant pas lancer les RPC, finalement après quelque recherche j'ai réussi à trouver les commandes les autorisant :

sudo -i service rpcbind stop
sudo -i rpcbind -i -w
sudo -i service rpcbind start

CODE TP3.X

```
/*service*/
typedef struct cellule* liste;
const MAXNOM = 255;
typedef string type_nom<MAXNOM>;
struct cellule{
        type_nom donnee;
        liste suivant;};
struct arg_writte{
        type_nom nom_fichier;
        int ecraser;
        liste * donnees;};
program TP3{
        version TP31{
                liste LS(char nom_repertoire)=1;
                liste READ(char nom_repertoire)=2;
                void WRITTE(arg_writte arg)=3;
        }=1;
}=0x20000088;
```

CODE REXO.C

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "tp3.h"
main (int argc,char *argv[]){
        CLIENT *cl;
        char *server;
        char nom_repertoire[256];
        liste *result;
        liste *result_read;
        liste *clone;
        int choix;
        bool_t again=TRUE;
        arg_writte *args;
        if (argc!=2) {
                fprintf(stderr, "usage: %s <host> \n", argv[0]);
                exit(1);
        }
        server = argv[1];
        /* creation de la poignee client */
        cl = clnt_create(server, TP3, TP31, "tcp");
        if (cl==NULL) {
                /* impossible d'etablir la connexion sur le serveur,
                impression du message d'erreur et fin.*/
                clnt_pcreateerror(server);
                exit(1);
        }
```

```
/* Sélectionner une identification */
cl->cl_auth = authunix_create_default();

/* Modification du timeout */
struct timeval delai;

delai.tv_sec = 60; /* nombre de secondes */
delai.tv_usec = 0; /* nombre de microsecondes */
clnt_control(cl, CLSET_TIMEOUT, (char *)&delai);

while (again){
        //code du menu//
        printf("\n");
        printf("*MENU TP3************\n");
        printf("*rentrer votre choix*\n");
        printf("**********************\n");
        printf("*1-LS              *\n");
        printf("*2-READ             *\n");
        printf("*3-WRITTE           *\n");
        printf("*4-EXIT            *\n");
        printf("**********************\n");
        printf("\nsaisir votre choix (1/2/3/4):\n");
        scanf("%d", &choix);
```

```
/* Appel de la procedure distante sur le serveur. */
switch (choix) {
	case 1:
		printf("saisir le nom du repertoire:\n");
		scanf("%s",&nom_repertoire);
		result=ls_1(nom_repertoire, cl);
		if (*result==NULL) {
			/* une erreur a eu lieu lors de l'appel du
serveur. Impression
			du message d'erreur et fin. */
			clnt_perror(cl, server);
			exit(1);
		}
		/* traitement du resultat  */
		printf("Liste recuperee \n");

		clone=result;
		while(*result !=NULL){

			printf("\n %s", (*result)->donnee);

			*result=(*result)->suivant;
		}
		xdr_free((xdrproc_t)xdr_liste,(char*)clone);

		break;
```

```
case 2:
        printf("saisir le nom du fichier:\n");
        scanf("%s",&nom_repertoire);

        result_read=read_1(nom_repertoire, cl);

        /*clone=result_read;
        while(*result_read !=NULL){
                printf("\n %s", (*result_read)->donnee);

                *result=(*result_read)->suivant;
        }*/
        xdr_free((xdrproc_t)xdr_liste,(char*)clone);

        break;
```

```
                case 3:
                        args=malloc(sizeof(arg_writte));
                        printf("saisir le nom du fichier:\n");
                        scanf("%s",&nom_repertoire);
                        strcpy(args->nom_fichier,nom_repertoire);
                        printf("Nom du fichier dans lequel write écrit :
%s\n",args->nom_fichier);
                                printf("ecraser les donnees? (1/0):\n");
                                scanf("%d", &choix);
                                args->ecraser=choix;
                        /* récupération du fichier de données par un read */
                        args->donnees = malloc(sizeof(cellule));
                        args->donnees = read_1("./lol", cl);
                        if (args->donnees==NULL)
                        {
                         /* une erreur a eu lieu lors de l'appel du serveur.
                          * Impression du message d'erreur et fin.
                          */
                         clnt_perror(cl, server);
                         exit(1);
                    }
                        result=writte_1(args, cl);
                        break;

                case 4:
                        again=FALSE;
                        break;
```

```
                default:
                        printf("mauvaise entree \n");
                }

        choix=0;
        }
    clnt_perror(cl,"127.0.0.1");
    printf("\n");
    exit(0);
}
```

CODE TP3_PROC.C

```
#include <stdio.h>
#include <rpc/rpc.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include "tp3.h"
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
/* version distante des procédures          *

* Rmq1 : toute procedure distante accepte  *
 * en entree un pointeur sur les arguments  *
 * qu'elle aurait eus en local              *

 * Rmq2 : toute procedure distante retourne *
 * un pointeur sur les resultats qu'elle    *
 * aurait retournes en local                *

 * Rmq3 : rpcgen convertit en minuscule le  *
 * nom de la procedure dans la definition   *
 * de programme, rajoute le caractere de    *
 * soulignement (_) et le numero de version */
```

```
/***************************************
 * Procédure distante ls_1_svc          *
 ***************************************/
#define SIZE 1024

extern liste *ls_1_svc(char *nom_repertoire, struct svc_req *rqstp){

        // On teste d'abbord l'authentification
        if ( rqstp->rq_cred.oa_flavor == AUTH_NULL )
        {
                printf("Aucune identification n'est utilisee.\n");
        }
        else if ( rqstp->rq_cred.oa_flavor == AUTH_UNIX )
        {
                printf("Type d'identification : UNIX.\n");

                // On déclare une structure
                struct authunix_parms* aup;

                // On récupère les paramètres d'identification
                aup = (struct authunix_parms*)rqstp->rq_clntcred;
                if ( (aup->aup_uid == getuid()) /* test de l'uid */ && (aup-
>aup_gid == getgid())/* test du groupe */ )
                {
                        printf("Authentification reussie.\n");
                }
                else
                {
```

```
                printf("Echec de l'authentification.\n");
        }
}
else
{
        printf("Erreur dans la reconnaissance de l'authentification.\n");
}
struct dirent * current;
DIR* ptdir;
int nb_result=0;

static liste n_f;
n_f = malloc(sizeof(cellule));
liste last=NULL;

// on ouvre le répertoire
ptdir=opendir(nom_repertoire);
if (!ptdir)
{
        fprintf(stderr,"error opendir\n");
}
// On lit le premier nom de dossier
current = readdir(ptdir);
if ( current != NULL )
{
        n_f->donnee = malloc(sizeof(char)*1024);
        strcpy(n_f->donnee, current->d_name);      }
```

```
        last = n_f;
        current = readdir(ptdir);
        // on parcourt les fichiers
        // pour chaque fichier ou dossier
        while(current != NULL)
        {
                last->suivant = malloc(sizeof(cellule));
                last = last->suivant;


                last->donnee =  malloc(sizeof(char)*1024);
                strcpy(last->donnee, current->d_name);
                nb_result+=1;
                printf("%s\n", last->donnee);
                printf("\n");
                current = readdir(ptdir);
        }
        closedir(ptdir);
        printf("%d",nb_result);
        printf("\n");
        // On vide le buffer
        fflush(stdout);
        fflush(stdin);
        return &n_f;


}
```

```
/*****************************************
 * Procédure distante read_1_svc        *
 *****************************************/
extern liste * read_1_svc(char *nom_repertoire, struct svc_req *rqstp){
        // On teste d'abbord l'authentification
        if ( rqstp->rq_cred.oa_flavor == AUTH_NULL )
        {
                printf("Aucune identification n'est utilisee.\n");
        }
        else if ( rqstp->rq_cred.oa_flavor == AUTH_UNIX )
        {
                printf("Type d'identification : UNIX.\n");
                // On déclare une structure
                struct authunix_parms* aup;

                // On récupère les paramètres d'identification
                aup = (struct authunix_parms*)rqstp->rq_clntcred;
                if ( (aup->aup_uid == getuid()) /* test de l'uid */ && (aup-
>aup_gid == getgid())/* test du groupe */ )
                {
                        printf("Authentification reussie.\n");
                }
                else
                {
                        printf("Echec de l'authentification.\n");
                }
        }
```

```
else
{
        printf("Erreur dans la reconnaissance de l'authentification.\n");
}
/*char *str = malloc(sizeof(char)*256);
strcpy(str,"lol.txt");*/

FILE* fichierParcouru= NULL;
// On ouvre le fichier
fichierParcouru = fopen(/*str/"lol.txt"*/nom_repertoire, "r");

//free(str);
// Gestion de l'erreur
if(fichierParcouru==NULL)
{
        perror("Erreur dans l'ouverture du fichier\n");
        exit(1);
}else{
        printf("ouverture reussie");
}

// Allouer la premiere cellule
static liste  n_f;
n_f= malloc(sizeof(cellule));
liste last=n_f;

last->donnee = malloc(sizeof(char)*1024);
```

```
last->suivant = NULL;
int count;
// Parcours du fichier
// pour chaque bloc de 1024 octets
char buffer[256];
bzero(buffer,256);
printf("\nVoici la liste des blocs lus par le read : \n\n\n");
do
{
	// On vide le buffer
	fflush(stdout);
	fflush(stdin);

	//récupérer un bloc de 1024 octets et le stocker dans la liste
	count=fread(buffer,1,1024,fichierParcouru);
	printf("nb de blocs  %d\n",count);
	if (count<=0){
		fprintf(stderr,"erreur lecture\n");
		exit(9);
	}
	printf("%s\n",last->donnee);
	bcopy(buffer,last->donnee,1024);
	// allouer la cellule
	last->suivant = malloc(sizeof(cellule));

	// passer au suivant
	last = last->suivant;
```

```
            last->suivant = NULL;
            last->donnee = malloc(sizeof(char)*256);
        }while(!feof(fichierParcouru));

        fclose(fichierParcouru);
        return &n_f;
}
```

```
/***************************************
 * Procédure distante write_1_svc       *
 ***************************************/
extern void * writte_1_svc(arg_writte *arg, struct svc_req *rqstp){
        // On teste d'abbord l'authentification
        if ( rqstp->rq_cred.oa_flavor == AUTH_NULL )
        {
                printf("Aucune identification n'est utilisee.\n");
        }
        else if ( rqstp->rq_cred.oa_flavor == AUTH_UNIX )
        {
                printf("Type d'identification : UNIX.\n");

                // On déclare une structure
                struct authunix_parms* aup;

                // On récupère les paramètres d'identification
                aup = (struct authunix_parms*)rqstp->rq_clntcred;
                if ( (aup->aup_uid == getuid()) /* test de l'uid */ && (aup-
>aup_gid == getgid())/* test du groupe */ )
                {
                        printf("Authentification reussie.\n");
                }
                else
                {
                        printf("Echec de l'authentification.\n");
                }
        }
```

```
else
{
        printf("Erreur dans la reconnaissance de l'authentification.\n");
}
FILE* fichier = NULL;
char mode[4];
static char retour = 'a' ;
if ( arg->ecraser == 1 )
{
        strcpy(mode,"w"); // mode d'écriture et purge
}
else
{
        strcpy(mode,"a"); // mode création et écriture
}
// On ouvre le fichier dans ce mode
fichier = fopen(arg->nom_fichier,mode);

// Gestion de l'erreur
if ( fichier == NULL )
{
        perror("Erreur dans l'ouverture du fichier.\n");

        // Sortie du programme
        retour = 'e';
        return (&retour);
}
```

```
        else
        {
                // On écrit dans le fichier
                liste courant = (*arg->donnees);

                // Parcours des données
                while ( courant != NULL )
                {
                        // On écrit les données dans le fichier
                        fprintf(fichier,"%s",courant->donnee);

                        // On passe au suivant
                        courant = courant->suivant;
                }
        }

        // On ferme le fichier
        fclose(fichier);

        // Sortie du programme
        return (&retour);
}
```