Gestion d’un index téléphonique par RPC

par Achraf EZZELDIN et Pierre LEVEQUE

L’objectif du TP est de développer un service RPC client/serveur pour gérer un index téléphonique. La première partie du TP consiste à écrire les fichiers et tester diverses fonctions, la seconde partie consiste à les générer en utilisant rpcgen.

Sommaire

TP1 : Gestion d'un index téléphonique par RPC

Introduction

L'objectif du TP est de développer un service RPC client/serveur pour gérer un index téléphonique. A travers une interface graphique, le client fera appel au serveur pour exécuter différente requête, telles que rechercher ou ajouter un numéro dans l'index, faire afficher un message par le serveur ou encore demander son arrêt à distance.
Le but du TP est de voir l'impact des services RPC haut niveau et bas niveau lors de la création de services client/serveur.

I/ Client / Serveur haut niveau

1) filtre XDR de la structure "ligne".

La structure ligne est définie dans le fichier index.h comme ceci :

```
struct ligne {
	char *nom;
	int tel;
};
typedef struct ligne ligne;
```

Son filtre XDR associé est défini dans le fichier index_xdr.c comme ceci :

```
#include "index.h"

bool_t xdr_ligne (XDR * flot, struct ligne * ptr){
	char ** nom = malloc(20 * sizeof(char));
	int tel;

	if (nom = NULL){
		printf("erreur malloc nom trop long !");
		exit(1);
	}
	return (xdr_string(flot, &ptr->nom, sizeof(nom))
		&& xdr_int(flot, &ptr->tel));
}
```

Ligne étant une structure, nous ne pouvons pas utiliser les filtres XDR standards (il faut en créer un spécifique : xdr_ligne). Cette structure se compose d'une chaîne de caractère (nom) et d'un entier (numéro de téléphone).
Notre filtre XDR customisé prend donc en second paramètre un pointeur sur une structure et utilise les filtres XDR standard (xdr_wrapstring et xdr_int) pour adapter les données au format XDR.
Nous allouons la mémoire pour éviter les fuites de mémoire.

2) Développement client/serveur RPC

À présent, nous allons créer les fichiers index_clnt.c et index_svc.c qui nous permettrons la transmission des flux sur le réseau via flots XDR mais avant nous devons modifier le fichier index.h en ajoutant les prototypes des fonctions utilisées.
Nous déclarons également le prototype de notre filtre XDR customisé “xdr_ligne”.

```
extern void * affiche_1();
extern ligne *recherche_1();
extern int * ajout_1();

extern bool_t xdr_ligne();
```

Nous précisons à l’aide de l’attribut “extern” que ces fonctions ne sont pas disponible en local au moment de la compilation mais que leurs localisation sera connue au moment de l'édition de liens.

Ces fonctions sont définie dans le fichier index_local.c :

```
#include "index.h"
#define NOM_FICHIER "index.data"

/* Recherche de la ligne contenant le nom correspondant... */

static ligne entree;
static char buf[256];

ligne * recherche_1 ( char ** nom ) {
  FILE *f;
  int t;

  entree.nom = (char *)buf; entree.tel = 0;
  if ((f = fopen (NOM_FICHIER, "r")) == NULL) {perror ("svc: fopen ");}
  else {
    while ( fscanf(f,"%s%d", buf, &t ) != EOF ) {
      if ( !strcmp( *nom, buf) ) {
      entree.tel = t;
      fclose ( f );
      return ( &entree );
      }
    }
    fclose ( f );
  } /* entry not found: return empty string */
  entree.nom[0] = '\0';
  return ( &entree );
}

/* Ajout d'une ligne si pas deja presente... */
```

```c
static int status_ajout;

int * ajout_1 ( ligne * a_ajouter ) {
  FILE *f;
  ligne *test;

  status_ajout = 0;
  test = recherche_1 ( &(a_ajouter->nom) );
  if ( ! (test->tel) ) {
    f = fopen ( NOM_FICHIER, "a" );
    fprintf (f, "%s %d\n", a_ajouter->nom, a_ajouter->tel);
      fclose(f);
      status_ajout = 1;
  }
  return ( &status_ajout );
}
/* Affiche une chaine de caractere... */

void * affiche_1 ( char ** s ) {
  printf ( "%s\n", *s ); fflush (stdout);
  return ((void *)1);
}
```

Retour au fichier index.h : nous définissons les alias pour les numéros de programme, version et les différents processus ainsi que les librairies nécessaires à l'utilisation du format XDR .
Pour le numéro de programme que nous allons utiliser, il faut qu'il soit compris dans la plage publique, à savoir dans l'intervalle [0x22222222 ; 0x3FFFFFFF] et il faut vérifier que les numéros est libre grâce à la commande « linux » suivante : $ rpcinfo –p.

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#include <rpc/rpc.h>
#include <rpc/clnt.h>
#include <rpc/svc.h>

#define NB_PROG 0x22222222
#define NB_VERS 1
#define PROC_RECH 1
#define PROC_ADD 2
```

Dans le ficher index_clnt.c nous utilisons la fonction de haut niveau callrpc().
Les« callrpc » correspondent à la demande d'exécution d'un processus sur une machine renseigné en paramètre avec un filtre d'entrée et un filtre de sortie. Le protocole de transport

est l'UDP et nous ne pouvons pas modifier certains paramètres, cela est dû à l'utilisation de l'API haut niveau.
Les données reçues devront être stockés dans des variables globales afin que le programme n'efface pas les données de la mémoire à la fin de l'exécution de la fonction.

```
#include "index.h"

struct ligne tab;
char ** nom;
int status_ajout;

ligne * recherche_1 ( char ** nom ) {
      callrpc("localhost", NB_PROG, NB_VERS, PROC_RECH, xdr_wrapstring,
nom, xdr_ligne, &tab);
      return(&tab);
}

int * ajout_1 ( ligne * a_ajouter ) {
      callrpc("localhost", NB_PROG, NB_VERS, PROC_ADD, xdr_ligne,
a_ajouter, xdr_int, &status_ajout);
      return(&status_ajout);
}
```

Coté serveur nous devons donc utiliser la fonction registerrpc(). Le fichier index_svc.c renseigne donc les différentes procédures sur le portmapper pour les identifier.

```
#include "index.h"

/* code serveur */
main (int argc, char * argv[]){

      char ** nom;
      struct ligne tab;

      registerrpc(NB_PROG,NB_VERS,PROC_RECH,
                  recherche_1,xdr_wrapstring,xdr_ligne);

      registerrpc(NB_PROG,NB_VERS,PROC_ADD,
                  ajout_1,xdr_ligne,xdr_int);


      svc_run();
}
```

rappel : Les numéros de programmes, de version et de procédures sont renseignés dans index.h qui est commun au client et au serveur. Ainsi, nous sommes sûrs de nous adresser à la bonne procédure vu que l'on utilise une seule et même dénomination.

La fonction svc_run() démarre le serveur. Le serveur devra donc se lancer via le programme "./index_svc" et attendra qu'on lui fasse une requete. Le client se lancera avec le programme "./index" qui affiche le menu des fonctions du serveur distant.
Le code source index.c est le suivant:

```
#include "index.h"

menu(){
  printf ( "\n       Menu:\n");
  printf ( "1) Rechercher un nom\n");
  printf ( "2) Ajouter un nom\n");
  printf ( "3) Faire afficher au srv\n");
  printf ( "4) lister les noms\n");
  printf ( "5) Terminer le serveur\n");
  printf ( "6) Recherche avec callback\n");
  printf ( "7) Connecter a un autre srv \n");
  printf ( "8) Quitter\n\n");
  printf ( "Faites votre choix + Enter: ");
}

main (int argc, char ** argv) {
  char *chaine = (char *)malloc(256), telephone[256];
  ligne *resultat, nouvelle;

  menu();
  while (  scanf("%255s", chaine) ) {
    switch ( chaine[0] ) {
      case '1': /* recherche une entree dans l'index */
      printf ( "Recherche nom : ");  scanf("%255s", chaine);
      resultat = recherche_1 ( &chaine );
      printf ( "Nom : %s \t tel : %d\n", resultat -> nom, resultat -> tel
);
      break;
      case '2': /* ajoute une entree  */
      printf ( "Ajout Nom : ");  scanf("%255s", chaine); nouvelle.nom =
chaine;
      printf ("Tel : "); scanf ("%d", &nouvelle.tel ); getchar();
      printf ("Ajout: %d \n", *(ajout_1( &nouvelle )) );
      break;
    case '3': /* affiche... */
      printf ( "Afficher : ");  scanf("%255s", chaine);
      printf ("%s\n", chaine);
      break;
    case '4': /* liste chainee - implémentée plus tard */
      break;
      case '5': /* terminer - implémentée plus tard */
      break;
    case '8': /* quitter le client */
      exit(0);
      break;
    default:
      printf ("Option non (encore) implémenté...");
      }
      menu();}}
```

3) Exemples d’utilisation des fonctions

Tout d’abord, nous renseignons quelques noms et numéros dans notre index téléphonique index.data :

```
Paul 0612345678
Pierre 0689784556
Achraf 0745895623
Toto 1234
Robert 0145788956
```

Ensuite, nous lançons le serveur avec la commande ./index.svc comme suit :

```
pc5008c:~/IN4R22.RPC/TP1> ./index_svc
```

Puis nous lançons le client avec ./index :

```
pc5008c:~/IN4R22.RPC/TP1> ./index

       Menu:
1) Rechercher un nom
2) Ajouter un nom
3) Faire afficher au srv
4) lister les noms
5) Terminer le serveur
6) Recherche avec callback
7) Connecter a un autre srv
8) Quiter

Faites votre choix + Enter:
```

Ci-suit un exemple d’utilisation des 3 fonctions implémentées “ajout”, puis “recherche” et enfin “liste” : nous commençons par ajouter le contact Michael dans l’index, puis nous le recherchons via son nom, enfin nous listons tous les contact de l’index.

```
pc5008c:~/IN4R22.RPC/TP1> ./index

       Menu:
1) Rechercher un nom
2) Ajouter un nom
3) Faire afficher au srv
4) lister les noms
5) Terminer le serveur
6) Recherche avec callback
7) Connecter a un autre srv
8) Quiter

Faites votre choix + Enter: 2
Ajout Nom : Michael
Tel : 0145788956
Ajout: 1

       Menu:
1) Rechercher un nom
2) Ajouter un nom
3) Faire afficher au srv
4) lister les noms
5) Terminer le serveur
6) Recherche avec callback
7) Connecter a un autre srv
8) Quiter

Faites votre choix + Enter: 1
Recherche nom : Michael
Nom : Michael    tel : 145788956

       Menu:
1) Rechercher un nom
2) Ajouter un nom
3) Faire afficher au srv
4) lister les noms
5) Terminer le serveur
6) Recherche avec callback
7) Connecter a un autre srv
8) Quiter

Faites votre choix + Enter: 4

Liste :

Paul 612345678
Pierre 689784556
Achraf 745895623
Toto 1234
Robert 145788956
Michael 145788956
```

II/ Fonction de listage

Nous voulons à présent ajouter une nouvelle fonction au serveur. Pour ce faire nous allons devoir modifier tous les fichiers déjà existant.

Pour commencer, nous devons déclarer dans le fichier index.h la structure liste, le numéro de processus, le filtre XDR associée à cette structure ainsi que la fonction qui sera située dans le fichier index_local.c :

```
#define PROC_CHAINE 3

struct liste {
      struct ligne tab;
      struct liste * next;
};
typedef struct liste liste;

extern liste * liste_1();

extern bool_t xdr_liste();
```

Ce filtre XDR customisé sera déclaré dans le fichier index_xdr.c, nous ajoutons donc ces lignes :

```
/* filtre XDR pour la struct liste */
bool_t xdr_liste(XDR * flot, liste * pl){
      return (
      /* encode la donnee du noeud */
      xdr_ligne(flot, &pl->tab) &&
      /* Si dernier noeud (pl->next == NULL), encore FALSE */
      /* Sinon encode TRUE et appel xdr-reference(flot, &pl->next, taille,
xdr_Liste */
      xdr_pointer(flot, (char **) &pl->next, sizeof(liste),
(xdrproc_t)xdr_liste)
    );
}
```

Le fichier index_local.c qui contiendra la fonction de listage. Nous ajoutons donc ces lignes :

```
liste * noeud;
liste * head;
liste * liste_1 () {
      FILE *f;
      int t,num=0;
      liste * lc;
      if ((f = fopen (NOM_FICHIER, "r")) == NULL) {
      perror ("svc: fopen ");
```

```
        }
        else {
        if(fscanf(f,"%s%d", buf, &t ) != EOF){
        noeud=malloc(sizeof(liste));
        head=noeud;
        noeud->tab.nom=strdup(buf);
        noeud->tab.tel = t;
        num++;
        }
        while ( fscanf(f,"%s%d", buf, &t ) != EOF ) {
        noeud->next=malloc(sizeof(liste));
        noeud=noeud->next;
        noeud->tab.nom=strdup(buf);
        noeud->tab.tel = t;
        num++;
        }
        noeud->next=NULL;
        fclose(f);
        }

        if (num==0){
        head = malloc(sizeof(liste));
        head->tab.nom = strdup("num non trouve");
        head->tab.tel = 0;
        head->next = NULL;
        }
        return (head);
}
```

Dans le fichier index_clnt.c nous devrons ajouter l'appel à la fonction avec son callrpc() comme ceci :

```
struct liste list;

liste * liste_1 () {
      callrpc("localhost", NB_PROG, NB_VERS, PROC_CHAINE, xdr_void, NULL,
xdr_liste, &list);
      return(&list);
}
```

Et dans le fichier index_svc.c le registerrc() correspondant :

```
    registerrpc(NB_PROG,NB_VERS,PROC_CHAINE,liste_1,xdr_void,xdr_liste);
```

Nous ajoutons au fichier index.c l'option numero 4 qui permet de lister l'annuaire :

```
    case '4': /* liste chainee */
      printf("\nListe : \n\n");
      retour = liste_1();
      while (retour !=NULL) {
            printf("%s %d\n", retour->tab.nom, retour->tab.tel);
```

```
            retour = retour->next;
      }
      break;
```

il faut également déclarer la liste * retour utilisée dans ce switch case

```
  liste *retour;
```

III / FONCTION D'ARRÊT DU SERVEUR

Ici nous avons choisi de montrer comment définir cette fonctionnalité en mode Haut Niveau. Dans le TP2, nous serons en Bas Niveau pour cette fonctionnalité.

Comme pour la fonction de listage, il faut ajouter plusieurs lignes à chaque fichier.

Pour commencer, nous devons déclarer dans le fichier index.h le numéro de processus ainsi que la fonction qui sera située dans le fichier index_local.c :

```
#define PROC_STOP 4

extern void * stop_1();
```

Le fichier index_local.c qui contiendra la fonction de listage. Nous ajoutons donc ces lignes :

```
/* terminer le serveur */
void * stop_1(char ** pass){
      char mdp[]="secret";
      if (!(strcmp(*pass,mdp))){ /* si le mot de passe ets incorrect */
      exit(EXIT_SUCCESS); /* terminaison du serveur */
      }
}
```

Dans le fichier index_clnt.c nous devrons ajouter l'appel à la fonction avec son callrpc() comme ceci :

```
void * stop_1 (char ** mdp){
      callrpc("localhost", NB_PROG, NB_VERS, PROC_STOP,
            xdr_wrapstring, mdp, xdr_void, NULL);
}
```

Et dans le fichier index_svc.c le registerrc() correspondant :

```
registerrpc(NB_PROG,NB_VERS,PROC_STOP,stop_1,xdr_wrapstring,xdr_void);
```

Nous ajoutons au fichier index.c l'option numéro 5 qui permet d'arrêter le serveur :

```
      case '5': /* terminer */
      printf("mot de passe :");
      scanf("%255s", str);
      stop_1(&str);
      break;
```

il faut également déclarer la liste * retour utilisée dans ce switch case

```
  char * str = malloc(10*sizeof(char));
```

TP 2 : Gestion d'un index téléphonique par RPC (rpcgen)

Introduction

L'objectif du TP est de développer un service RPC client/serveur pour gérer un index téléphonique. Contrairement au premier TP, ici les fichiers ne seront pas écrit à la main mais seront générer en utilisant rpcgen et le Makefile correspodant.
Le but du TP est de voir l'impact des services RPC haut niveau et bas niveau lors de la création de services client/serveur, et comment rpcgen modifie ces services.

I/ Remarques

Afin de ne pas écraser nos fichiers du TP 1, nous avons re-télécharger l'archive de base pour utiliser rpcgen car à partir d'un fichier.x, rpcgen créer tous les autres fichiers nécessaires.
De plus, nous avons modifié le Makefile de façon à compilé correctement en dé-commentant les lignes concernant rpcgen.

II/ Rpcgen

De manière à réaliser une application client/serveur en implémentant les services RPC de base de gestion d'annuaire téléphonique, nous avons écrit un fichier de spécifications RPCL nommé index.x :

```
const SIZE=50;

struct ligne {
      string nom <SIZE>;
      int tel;
};

struct liste {
      struct ligne tab;
      struct liste * next;
};

program NB_PROG{
      version NB_VERS{
            int AJOUT(ligne)=1;
            ligne RECHERCHE(string)=2;
            liste LISTE(void)=3;
            void STOP(string)=5;
```

```
        }=1;
}=0x22222222;
```

Afin de recréer les fonctions du TP1, nous avons créer les structures ligne et liste. De plus, nous avons renseigné le numéro de programme (0x22222222), le numéro de version (1) ainsi que les numéros des différentes options.

Lors de la compilation avec le Makefile, rpcgen créer automatiquement les autres fichiers nécessaires au fonctionnement des services RPC implémentés, à savoir rechercher et ajouter un numéro, faire afficher un message au serveur et demander son arrêt à distance.

Nous avons ajouté dans le fichier index.c :
clnt = clnt_create("localhost",NB_PROG,NB_VERS,"udp");
Cete commande nous permet de créer un client UDP qui fera les appels aux fonctions RPC via callrpc.

Après que rpcgen ai généré les nouveaux fichiers, nous modifions index.c, index_svc.c et index_local.c en ajoutant aux noms des fonctions la notion “_svc” ainsi qu’en ajouant à nos fonctions un second paramètre “CLIENT * clnt”. Cela permettra aux fonctions d’utiliser un client UDP pour faire des callrpc.

Par exemple, voici la définition de la fonction “ajout_1” dans le fichier index_clnt.c (en rouge ce qui a été ajouté) :

```
int * ajout_1(ligne *argp, CLIENT *clnt)
```

Un autre exemple, pour la même fonction mais cette fois-ci dans le fichier index_local.c :

```
int * ajout_1_svc ( ligne * a_ajouter, struct svc_req *clnt )
```

III/ Services complémentaires

1) liste de l’index

La fonction de listage ayant déjà été ecrite lors du précédent TP, nous avons seulement eu à la modifier en ajoutant le second paramètre “clnt”. Voici son cas correspondant dans le fichier index.c :

```
        case '4': /* liste chainee */
        sec=liste_1(NULL,clnt);
        prems=sec;
        while(sec->next !=NULL){
```

```
                printf("Nom: %s  Tel: %d\n", sec->tab.nom,sec->tab.tel);
                sec=sec->next;
        }
        printf("Nom: %s  Tel: %d\n", sec->tab.nom,sec->tab.tel);
        sec=prems->next;
        while(sec !=NULL){
                prems=sec->next;
                free(sec);
                sec=prems;
        }
```

Voici la fonction liste présente dans le fichier index_clnt.c :

```
liste *
liste_1(void *argp, CLIENT *clnt)
{
   static liste clnt_res;

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

Et maintenant la même fonction dans le fichier index_local.c

```
liste * noeud;
liste * head;
/* liste les lignes */
liste * liste_1_svc (void* toto, struct svc_req * clnt ) {
   FILE *f;
   int t,num=0;
   liste * lc;
   if ((f = fopen (NOM_FICHIER, "r")) == NULL) {
         perror ("svc: fopen ");
   }
   else {
   if(fscanf(f,"%s%d", buf, &t ) != EOF){
         noeud=malloc(sizeof(liste));
         head=noeud;
         noeud->tab.nom=strdup(buf);
         noeud->tab.tel = t;
         num++;
   }
```

```
   /*lc=&noeud;*/
   while ( fscanf(f,"%s%d", buf, &t ) != EOF ) {
         noeud->next=malloc(sizeof(liste));
         noeud=noeud->next;
         noeud->tab.nom=strdup(buf);
         noeud->tab.tel = t;
         num++;
   }
   noeud->next=NULL;
   fclose(f);
   }

   if (num==0){
         head = malloc(sizeof(liste));
         head->tab.nom = strdup("num non trouve");
         head->tab.tel = 0;
         head->next = NULL;
   }
   return (head);
}
```

Enfin, le filtre xdr_liste du fichier index_xdr.c n'a pas changé.

2) arrêt du serveur

Contrairement au précédent TP, nous avons fait notre fonction stop en bas niveau.
Comme pour la fonction liste, nous n'avons eu que peu de modifications à faire.

Cas dans le fichier index.c :

```
         case '5': /* terminer */
         printf("mot de passe :");
         scanf("%255s", str);
         stop_1(&str, clnt);
         break;
```

Stop dans le fichier index_clnt.c :

```
void *
stop_1(char **argp, CLIENT *clnt)
{
   static char clnt_res;

   memset((char *)&clnt_res, 0, sizeof(clnt_res));
   if (clnt_call (clnt, STOP,
         (xdrproc_t) xdr_wrapstring, (caddr_t) argp,
         (xdrproc_t) xdr_void, (caddr_t) &clnt_res,
         TIMEOUT) != RPC_SUCCESS) {
```

```
        return (NULL);
    }
    return ((void *)&clnt_res);
}
```

Stop dans le fihcier index_local.c :

```
/* terminer le serveur */
void * stop_1_svc(char ** pass, struct svc_req * clnt){
    char mdp[]="secret";
    if (!(strcmp(*pass,mdp))){ /* si le mot de passe ets incorrect */
        exit(EXIT_SUCCESS); /* terminaison du serveur */
    }
}
```

Dans le fichier index.x, nous avons spécifié void comme type de retour de stop.