Université Mohammed V – Rabat
Ecole Mohammadia d'Ingénieurs
Département Génie Informatique

TP N°1

Sujet

RPC

Réalisé par :
BENHLIMA Douae

Encadré par :
Mme. CHIADMI DALILA

ANNÉE UNIVERSITAIRE 2022-2023

Table des matières

1 Introduction

1.1 Description de l'Appel de Procédure à Distance (RPC - Remote Procedure Call) :

L'Appel de Procédure à Distance (RPC - Remote Procedure Call) est une méthode de communication interprocessus qui permet à un programme d'exécuter un code sur une machine distante sans se soucier explicitement des détails de la communication en réseau. Dans un système RPC, les fonctions sont exécutées sur un serveur, tandis que les clients peuvent appeler ces fonctions comme si elles étaient exécutées localement. Cette abstraction simplifie grandement le développement d'applications distribuées, car elle masque les complexités de la gestion des sockets, de la sérialisation des données et de la gestion des adresses de réseau.

1.2 Rôle de RPC dans la communication entre différents systèmes d'exploitation et applications dans un environnement distribué :

RPC joue un rôle crucial dans la communication entre différents systèmes d'exploitation et applications dans un environnement distribué. Il fournit un mécanisme par lequel un processus (un client) peut demander à un autre processus (un serveur) d'exécuter une certaine tâche et de renvoyer le résultat. Cela permet aux systèmes d'exploitation et aux applications de collaborer de manière transparente sur différents ordinateurs et réseaux, facilitant ainsi le partage de ressources et la répartition des charges de travail.

1.3 Nécessité de l'utilisation de RPC dans les systèmes distribués :

Dans un système distribué, les tâches sont réparties entre plusieurs ordinateurs qui travaillent ensemble pour atteindre un objectif commun. RPC est essentiel dans de tels systèmes car il permet la communication et la collaboration entre ces ordinateurs. Grâce à RPC, un ordinateur (client) peut demander à un autre ordinateur (serveur) d'exécuter une tâche spécifique. Cela permet de distribuer la charge de travail et d'exploiter la puissance de traitement de plusieurs machines. De plus, RPC facilite la répartition de la charge de travail en permettant à chaque machine de se concentrer sur ce qu'elle fait le mieux.

2 Description du code du serveur :

2.1 Inclusion des bibliothèques nécessaires

```c
#include <stdio.h>
#include <rpc/rpc.h>
```

Dans cette section, les bibliothèques standard nécessaires sont incluses. "stdio.h" est utilisé pour les opérations d'entrée/sortie standard et "rpc/rpc.h" est la bibliothèque standard pour les fonctions RPC.

2.2 Définition des constantes

```c
// définir un programme, une version et une procédure
#define PROGRAMME ((unsigned long)0x33333333)
#define VERSION ((unsigned long)1)
#define PROCEDURE ((unsigned long)1)
```

Ici, on définit des constantes pour le numéro du programme, la version et le numéro de la procédure. Ces constantes sont utilisées pour identifier la fonction spécifique qui sera enregistrée et appelée via RPC.

2.3 Définition de la fonction à enregistrer

```c
// la fonction à enregistrer
char *echo(char *arg) {
    return arg;
}
```

Ici, on définit la fonction qui sera appelée par le client. Dans cet exemple, il s'agit d'une simple fonction "echo" qui renvoie ce qu'elle reçoit en argument.

2.4 Fonction main - Enregistrement de la fonction et démarrage du serveur

```c
int main(void) {
    // enregistrer la fonction echo
    if (registerrpc(PROGRAMME, VERSION, PROCEDURE, echo, (xdrproc_t)xdr_wrapstring, (xdrproc_t)xdr_wrapstring) == -1) {
        fprintf(stderr, "Erreur d'enregistrement de la RPC\n");
        exit(1);
    }

    svc_run(); // commencer à écouter les requêtes RPC
    fprintf(stderr, "Erreur: svc_run a retourné\n");
    exit(1);
}
```

La fonction main effectue deux opérations importantes.

Elle commence par enregistrer la fonction "echo" avec le service RPC, en utilisant le numéro de programme, le numéro de version et le numéro de procédure définis précédemment. La fonction "echo" est enregistrée avec deux fonctions de sérialisation/désérialisation (xdr_wrapstring dans cet exemple), qui sont utilisées pour convertir les données entre leur représentation interne et un format standard qui peut être transmis sur le réseau. Si l'enregistrement échoue, un message d'erreur est affiché et le programme se termine.

Ensuite, elle appelle la fonction svc_run, qui fait entrer le serveur dans une boucle où il attend et traite les appels RPC des clients. Si svc_run revient, cela signifie qu'une erreur s'est produite, donc un message d'erreur est affiché et le programme se termine.

3 Description du code du client

3.1 Inclusion des bibliothèques nécessaires

```c
client.c > ...
1   #include <stdio.h>
2   #include <rpc/rpc.h>
```

Tout comme dans le code du serveur, on commence par inclure les bibliothèques nécessaires. "stdio.h" est utilisé pour les opérations d'entrée/sortie standard et "rpc/rpc.h" contient toutes les fonctions RPC.

3.2 Définition des constantes

```c
4   // les mêmes définitions que dans le serveur
5   #define PROGRAMME ((unsigned long)0x33333333)
6   #define VERSION ((unsigned long)1)
7   #define PROCEDURE ((unsigned long)1)
8
```

Ici, on définit des constantes pour le numéro du programme, la version et le numéro de la procédure. Ces constantes doivent correspondre à celles définies dans le serveur afin que le client puisse appeler la bonne fonction.

3.3 Fonction main - Appel de la fonction distante et affichage du résultat

```c
9   int main(int argc, char *argv[]) {
10      char *server;
11      char *message;
12      char **result;
13      enum clnt_stat stat;
14
15      if (argc != 3) {
16          fprintf(stderr, "Utilisation: %s serveur message\n", argv[0]);
17          exit(1);
18      }
19      server = argv[1];
20      message = argv[2];
```

```c
    // Allocate memory for result
    result = (char **)malloc(sizeof(char *));
    if(result == NULL) {
        fprintf(stderr, "Failed to allocate memory\n");
        exit(1);
    }
    *result = NULL; // ensure it's initialized to NULL

    // Call RPC
    stat = callrpc(server, PROGRAMME, VERSION, PROCEDURE, (xdrproc_t)xdr_wrapstring, &message, (xdrproc_t)xdr_wrapstring, result);
    if (stat != RPC_SUCCESS) {
        clnt_perrno(stat);
        exit(1);
    }
    if (*result == NULL) {
        printf("Pas de réponse du serveur\n");
    } else {
        printf("Réponse du serveur: %s\n", *result);
    }
    free(result);  // don't forget to free the allocated memory
    exit(0);
}
```

La fonction main du client est un peu plus complexe que celle du serveur. Elle commence par vérifier que le bon nombre d'arguments a été fourni (le nom du serveur et le message à envoyer). Elle stocke ensuite ces arguments dans des variables pour une utilisation ultérieure.

Ensuite, elle alloue de la mémoire pour stocker le résultat de l'appel RPC.

La fonction callrpc est utilisée pour effectuer l'appel RPC. Cette fonction prend en arguments le nom du serveur, le numéro de programme, la version, le numéro de procédure, les fonctions de sérialisation/désérialisation, l'argument à passer à la fonction distante, et un pointeur vers un endroit où stocker le résultat.

Si l'appel RPC échoue, callrpc renvoie un code d'erreur et un message d'erreur est affiché.

Enfin, le résultat de l'appel RPC est affiché. Si aucun résultat n'a été reçu du serveur, un message est affiché pour indiquer cela. Sinon, le résultat est affiché.

À la fin, la mémoire allouée pour le résultat est libérée.

4 Démonstration

Pour la compilation des deux programmes il faut utiliser les commandes suivantes :

gcc -I/usr/include/tirpc -o server serveur.c -ltirpc

gcc -I/usr/include/tirpc -o client client.c -ltirpc

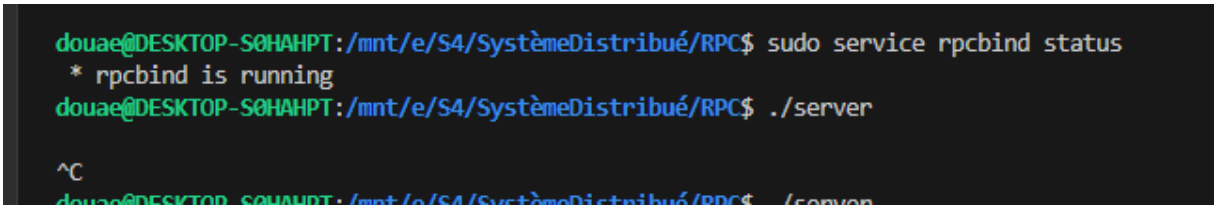
```
douae@DESKTOP-S0HAHPT:/mnt/e/S4/SystèmeDistribué/RPC$ gcc -I/usr/include/tirpc -o server serveur.c -ltirpc
douae@DESKTOP-S0HAHPT:/mnt/e/S4/SystèmeDistribué/RPC$ gcc -I/usr/include/tirpc -o client client.c -ltirpc
douae@DESKTOP-S0HAHPT:/mnt/e/S4/SystèmeDistribué/RPC$
```

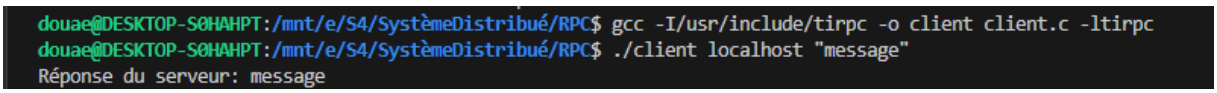
Premièrement il faut lancer le serveur le premier :

```
douae@DESKTOP-S0HAHPT:/mnt/e/S4/SystèmeDistribué/RPC$ ./server
```

Donc, dans le contexte de ce programme RPC, le serveur doit être lancé en premier pour qu'il puisse enregistrer la procédure RPC (echo) avec le système d'exploitation. Une fois que le serveur a enregistré la procédure, le client peut alors l'appeler. Si le client est lancé en premier et tente d'appeler une procédure qui n'a pas encore été enregistrée, l'appel échouera.

Le serveur se met en mode écoute et attend les requêtes des clients :



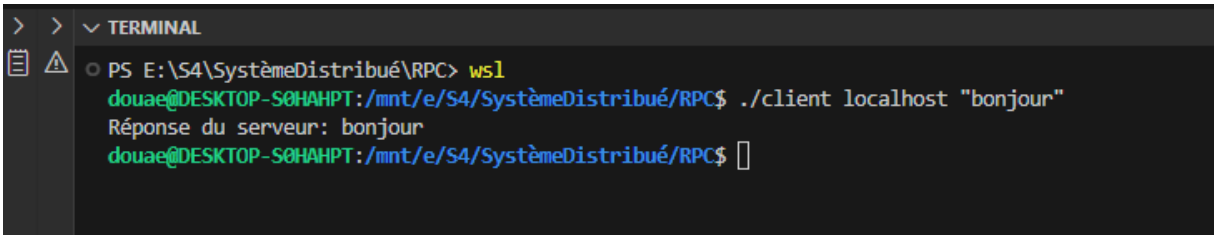


Puisque le programme client sur la même machine que le serveur donc on peut utiliser localhost

Le client envoie la chaîne "Bonjour" au serveur

Le serveur reçoit la chaîne "Bonjour", l'envoie de retour au client (car la procédure echo renvoie simplement la chaîne qu'elle reçoit) et continue d'écouter d'autres requêtes.

Le client reçoit la réponse du serveur, qui doit être la même chaîne "Bonjour", et l'affiche.



5 Conclusion

Synthèse des points clés de l'explication du code

Nous avons passé en revue le code du client et du serveur pour une application utilisant RPC (Remote Procedure Call). Ces programmes illustrent comment une fonction peut être enregistrée sur un serveur et appelée par un client, même si les deux sont exécutés sur des systèmes distincts dans un environnement distribué.

Le code du serveur définit et enregistre une fonction qui peut être appelée à distance. Cette fonction est ensuite enregistrée à l'aide de registerrpc, avec les identifiants appropriés (programme, version et numéro de procédure).

Le code client effectue l'appel de la procédure à distance. Il utilise callrpc, en passant le nom du serveur et les identifiants appropriés, ainsi que les arguments pour la fonction à appeler. Il reçoit ensuite le résultat et l'affiche.

Discussion sur l'importance de la gestion correcte des erreurs et de la libération de la mémoire dans le code du client

La gestion des erreurs est essentielle dans toute application, mais elle est particulièrement importante lors de l'utilisation de RPC. En raison de la nature distribuée de l'environnement, de nombreuses choses peuvent mal tourner : la fonction demandée peut ne pas exister sur le serveur, le serveur peut être hors ligne ou inaccessible, etc. Notre code client doit donc vérifier le statut de chaque appel RPC et gérer correctement les erreurs.

La gestion de la mémoire est également cruciale. Dans notre code client, nous avons alloué de la mémoire pour stocker le résultat de l'appel RPC. Il est de notre responsabilité de libérer cette mémoire une fois que nous avons terminé avec elle, pour éviter les fuites de mémoire qui pourraient éventuellement épuiser toutes les ressources du système.

Réflexion sur les avantages et les défis de l'utilisation de RPC dans les systèmes distribués

L'utilisation de RPC présente de nombreux avantages. Elle permet une séparation claire des responsabilités entre le client et le serveur, rend possible la création de systèmes distribués plus complexes, et permet aux composants d'un système de communiquer de manière transparente, malgré les différences d'architecture et de localisation.

Cependant, elle présente également des défis. La gestion des erreurs peut être complexe, en raison de la variété des problèmes qui peuvent survenir. Les problèmes de performance peuvent également se poser, en particulier lors de l'appel de fonctions sur des réseaux lents. De plus, les problèmes de sécurité doivent être pris en compte, car les appels RPC peuvent être interceptés par des acteurs malveillants.