

# Tutoriel **rpcbind** sur Linux

Pour ce tutoriel il est nécessaire de démarrer d'abord Linux en utilisant votre machine virtuelle. Pour cela, consulter le tuto précédent.

## Partie I : Installation de **rpcbind** sur Linux

### Vérification/Installation de **rpcbind** sur Linux

Ouvrez un terminal. Tapez cette commande ci-dessous :

```
$ rpcinfo
```

S'il y est installé, alors voici à peu près un aperçu de ce que vous devez avoir sur votre terminal.

```
b-diop@bdtop-Aspire-E1-570:~$ rpcinfo
program version netid address service owner
100000 4 tcp6 :::.0.111 portmapper superuser
100000 3 tcp6 :::.0.111 portmapper superuser
100000 4 udp6 :::.0.111 portmapper superuser
100000 3 udp6 :::.0.111 portmapper superuser
100000 4 tcp 0.0.0.0.0.111 portmapper superuser
100000 3 tcp 0.0.0.0.0.111 portmapper superuser
100000 2 tcp 0.0.0.0.0.111 portmapper superuser
100000 4 udp 0.0.0.0.0.111 portmapper superuser
100000 3 udp 0.0.0.0.0.111 portmapper superuser
100000 2 udp 0.0.0.0.0.111 portmapper superuser
100000 4 local /run/rpcbind.sock portmapper superuser
100000 3 local /run/rpcbind.sock portmapper superuser
b-diop@bdtop-Aspire-E1-570:~$
```

Au cas où **rpcbind** n'est pas installé, tapez les commandes ci-dessous. Ceux-ci permettent de mettre à jour les paquets logiciels et d'installer **rpcbind**.

```
$ sudo apt-get update
$ sudo apt-get install rpcbind
```

Retapez la commande **\$ rpcinfo** afin de vérifier si tout se passe bien.

### Tâches :

Dans ce qui suit, nous allons créer un mini-projet client-serveur, qui va effectuer l'addition de deux nombres.

Créer un répertoire dans le dossier /home. Pour cela, exécuter les commandes ci-dessous :

- Revenir au répertoire /home :  
\$ cd
- Cette commande va créer un répertoire nommé « newrpc ».  
\$ sudo mkdir newrpc  
\$ cd newrpc

- Créez un fichier dans le répertoire newrpc.  
\$ gedit add.x

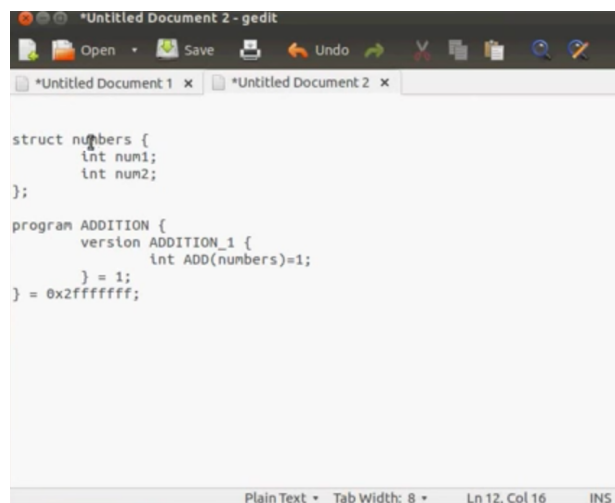
Voici un aperçu de ce que vous devez avoir. Le fichier add.x est fourni en pièce jointe de la page. Télécharger le fichier et partagez le avec votre machine virtuelle.

Sinon vous pouvez éditer un fichier add.x avec le contenu ci-dessous :

```
struct numbers {           # il y a un espace après
« numbers »
    int num1;              # Tabulation avant le int
    int num2;              # Tabulation avant le int
};

program ADDITION {        # il y a un espace après
« ADDITION»
    version ADDITION_1 { #il y a un espace après « ADDITION_1»
        int ADD(numbers)=1;
    } = 1;                 # il y a un espace avant et après
« = »
} = 0x2fffffff;           # il y a un espace avant et après
« = »
```

Avant de copier ces quelques lignes, noter qu'il est important de respecter l'indentation (l'espace entre les mots) du fichier add.x.



### Compilation du fichier add.x

Avant d'effectuer la compilation, assurez vous que vous êtes dans le répertoire newrpc.

Tapez la commande \$ pwd afin de vérifier le répertoire courant.

Si vous n'y êtes pas, tapez ces deux commandes :

```
$ cd
$ cd newrpc
```

Si tout se passe, tapez la commande de compilation du fichier add.x

```
$ rpcgen -a -C add.x
```

L'option C (C majuscule) indique à rpcgen de générer du code C qui est conforme à la norme standard ANSI C. Ce qui est la valeur par défaut sur certaines versions de rpcgen. Si nous regardons les fichiers qui ont été générés, nous allons voir:

**add.h** Ceci est le fichier d'en-tête que nous allons inclure à la fois sur le code client et le code serveur. Il définit la structure que nous avons défini (intpair) et le renomme à un type du même nom. Il définit également les symboles ADD\_PROG (0x23451111, notre numéro de programme) et ADD\_VERS (1, notre numéro de version). Ensuite, il définit l'interface stub client (ADD\_1) et l'interface de la fonction côté serveur.

**add\_server.c** : Ceci est le fichier add\_svc.c, le programme serveur. La fonction nommée add\_prog\_1 (le \_1 est utilisée pour identifier le numéro de version. La fonction contient une signature pour toutes les procédures à distance supportées par ce programme et cette version. En plus de la procédure null (qui est toujours prise en charge), la seule entrée dans l'instruction switch est ADD, pour notre fonction add. Ceci définit un pointeur de fonction (locale) à la fonction de serveur, **add\_1\_svc**. Plus tard dans la procédure, la fonction est appelée avec le paramètre « démarshalées » et l'information du demandeur.

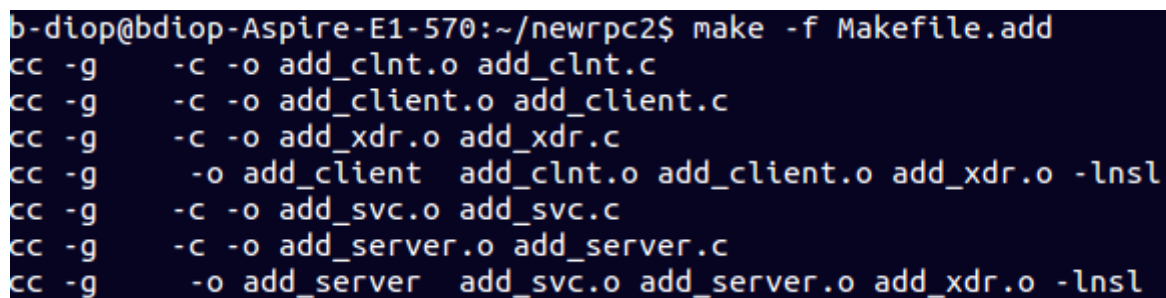
**add\_client.c** Ceci est fonction de stub client qui implémente la fonction ADD\_1. Il mobilise le paramètre, appelle la procédure à distance, et renvoie le résultat.

**add\_xdr.c** Le fichier \_xdr.c n'est pas toujours généré; cela dépend des paramètres utilisés pour les procédures à distance. Ce fichier contient le code pour rassembler les paramètres pour la structure intpair. Il utilise les bibliothèques XDR (eXternal Data Representation) pour convertir les deux entiers en un format standard.

Une fois les fichiers client et serveur générés, il est nécessaire de les compiler afin d'obtenir les exécutables (client et serveur). La compilation se fait en utilisant la commande make, comme suit :

```
$ make -f Makefile.add
```

La compilation génère un ensemble de fichiers, y compris les fichiers exécutables (add\_client, add\_server).



```
b-diop@bdiop-Aspire-E1-570:~/newrpc2$ make -f Makefile.add
cc -g -c -o add_clnt.o add_clnt.c
cc -g -c -o add_client.o add_client.c
cc -g -c -o add_xdr.o add_xdr.c
cc -g -o add_client add_clnt.o add_client.o add_xdr.o -lnsl
cc -g -c -o add_svc.o add_svc.c
cc -g -c -o add_server.o add_server.c
cc -g -o add_server add_svc.o add_server.o add_xdr.o -lnsl
```

Les fichiers add\_client et add\_server seront exécutés séparément sur deux terminaux.

Commençons d'abord par exécuter le programme serveur (add\_server) ensuite celui du client (add\_client), comme suit :

Premier terminal :

```
$ sudo ./add_server
```

Second terminal :

```
$ sudo ./add_client
```

Après cette première compilation, si tout se passe bien, aucun message n'est affiché à l'écran. Si ce n'est un message d'erreur, au cas où la compilation ne s'est pas bien déroulée.

## **Partie II : Application**

Dans cette partie, nous allons modifier les fichiers client et serveur afin que notre opération d'addition soit effective. Assurez vous que vous êtes positionné sur le répertoire contenant le fichier `add_client.c` et `add_server.c` ; le même que vous aviez créé dans la Partie I.

A présent, il faut faire faire au serveur et au client des tâches à accomplir. Pour cela, les fichiers `add_client.c` et `add_server.c` seront modifiés. Le client pourra dès lors envoyer au serveur deux entiers en utilisant la structure créée dans le fichier [add.x](#). Ensuite, une recompilation sera requise et les fichiers exécutables seront recréés.

### **Modification du fichier `add_client.c`**

Ouvrez le fichier `add_client.c` avec un éditeur de votre choix. Voici un exemple d'aperçu du fichier ouvert avec la commande `$ nano add_client.c`.

```

/*
 * 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 "add.h"

void
add_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    intpair add_1_arg;

#ifdef DEBUG
    clnt = clnt_create (host, ADD_PROG, ADD_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */
    add_1_arg.a = 123;
    add_1_arg.b = 100;
    result_1 = add_1(&add_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    else {
        printf("resultat addition %d \n", *result_1);
    }
#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}

```

#### Aperçu du fichier add\_client.c (1)

```

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

    if (argc < 2) {
        printf ("usage: %s server_host\n", argv[0]);
        exit (1);
    }
    host = argv[1];
    add_prog_1 (host);
    exit (0);
}

```

#### Aperçu du fichier add\_client.c (2)

```

/*
 * 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 "add.h"

int *
add_1_svc(intpair *argp, struct svc_req *rqstp)
{
    static int result;

    printf(" Addition de deux entiers !\n");
    printf(" Paramètres : %d, %d \n", argp->a, argp->b);
    result = argp->a + argp->b;
    printf(" Resultat = %d\n", result);
    return &result;
}

```

### Aperçu du fichier add\_server.c

Une fois le fichier add\_client ouvert, remplacer son contenu par le nouveau contenu du fichier add\_client.c ci-dessous. Refaites la même chose pour le fichier add\_server.c.

### Contenu du fichier add\_client.c

```

/*
 * 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 "add.h"

void
add_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    intpair add_1_arg;

#ifdef DEBUG
    clnt = clnt_create (host, ADD_PROG, ADD_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */
    add_1_arg.a = 123;
    add_1_arg.b = 100;
    result_1 = add_1(&add_1_arg, clnt);
    if (result_1 == (int *) NULL) {

```

```

        clnt_perror (clnt, "call failed");
    }
    else {
        printf("resultat addition %d \n", *result_1);
    }
#endif
#ifdef 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];
    add_prog_1 (host);
    exit (0);
}

```

### Contenu du fichier add\_server.c

```

/*
 * 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 "add.h"

int *
add_1_svc(intpair *argp, struct svc_req *rqstp)
{
    static int result;

    printf(" Addition de deux entiers !\n");
    printf(" Paramètres : %d, %d \n", argp->a, argp->b);
    result = argp->a + argp->b;
    printf(" Resultat = %d\n", result);
    return &result;
}

```

### Recompilation des fichiers

Positionnez-vous sous le répertoire newrpc et tapez cette commande pour recompiler les fichiers.

```
$ make -f Makefile
```

Si tout se passe bien, les fichiers exécutables seront reconstruits et vous pouvez les tester sur deux terminaux, en exécutant d'abord le serveur, ensuite le client.

Sur un premier terminal

```
$ ./add_server
```

Ensuite sur un second terminal

```
$ ./add_client localhost
```

Au cas où des erreurs sont générées durant la recompilation, repérer le bug en lisant la documentation. Ensuite rééditer le fichier concerné. Vérifier si le contenu de votre fichier (add\_client.c ou add\_server.c) est conforme avec les contenus décrits dans la section précédente.