Notes sur le protocole RPC

Bruno Jacob, LIUM, M1

19 Décembre 2003

Ce protocole à été défini chez Sun en vue de l'implémentation du système de fichiers répartis NFS. Le concept général de RPC (*Remote Procedure Call*) est le même que celui d'un appel d'une procédure locale, mais celle-ci s'exécute sur une autre machine. Le principe est donc de passer des paramètres et de récupérer une valeur de retour, via un réseau avec le protocole XDR, à une procédure ou fonction distante.

La demande d'appel d'une procédure distante est prise en compte par un programme démon qui tourne sur la machine sur laquelle doit s'exécuter la procédure.

Le protocole RPC est indépendant
- des protocoles sous-jacents utilisés pour l'échange des paramètres
- du système d'exploitation utilisé

Dans son utilisation sous Unix, le protocole sous-jacent est UDP pour la majeure partie des applications (par exemple NFS) et plus rarement TCP.

Le principe général du protocole RPC est montré dans la figure 1.

1 Les grandes lignes du protocole

Le protocole doit permettre principalement
- au serveur, d'enregistrer la procédure RPC pour qu'elle soit "visible" des clients
- aux clients, d'appeler la procédure RPC

1.1 Identification des procédures

Les procédures sont regroupées en programme (= 1 service spécifique). Pour identifier les différentes versions d'un programme et les procédures

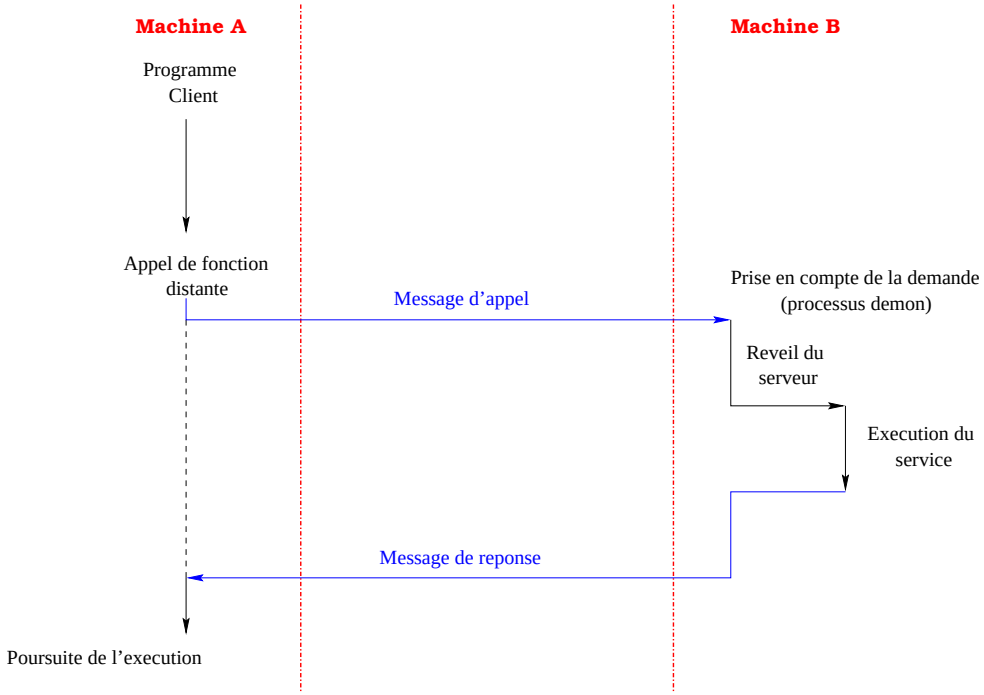

FIG. 1 – Principe général du protocole RPC

qu'elles contiennent on utilise des numéros.

Ainsi, pour identifier une procédure distante, il faut préciser :
- le numéro de programme
- le numéro de la version du programme
- le numéro de procédure dans le programme

Un démon sur la machine serveur se charge de prendre ne compte une demande de procédure ainsi identifiée.

1.2 Les numéros de programmes

Comme pour les numéros de ports correspondant à des services UDP ou TCP, certaines valeurs sont réservées. Le tableau 1 donnent les groupes des valeurs utilisables et réservées.

Intervalle hexadécimal	Intervalle décimal	Usage
[0x00000000..0x1fffffff]	[0..512M-1]	réservé
[0x20000000..0x3fffffff]	[512M..1G-1]	non réservé
[0x40000000..0xffffffff]	[1G..4G-1]	réservé

TAB. 1 – Numéros des programmes dans le protocole RPC

2 L'implémentation sous Unix

On peut distinguer les étapes suivantes pour créer des procédures distantes :

1. l'écriture du service proprement dit
2. l'activation du service
3. l'enregistrement du service

2.1 L'écriture du service

Elle revient à écrire les fonctions de traitement du service proprement dit et celles de sérialisation et de désérialisation des informations transmises en utilisant le protocole XDR.

2.2 L'activation du service

L'activation d'une procédure distante est réalisée par un processus de service. Ce processus communique avec le processus demandeur par une socket dans le domaine Internet avec le protocole UDP ou TCP. Une telle socket est associée à un numéro de port du protocole correspondant.

Si le concepteur du service est un utilisateur standard du système, alors il peut mettre le processus de service en écoute des appels de procédures distantes sur sa socket. Si c'est le super-utilisateur qui réalise ce service, il peut alors mettre la socket d'écoute dans un ensemble de ports. Cet ensemble est scruté par un processus particulier (le processus `idnet`) qui lui, crée le processus de service quand une demande arrive sur la socket d'écoute.

2.3 L'enregistrement du service

L'enregistrement du service sur la machine du serveur est fait par le processus `portmap` qui est lui-même un service RPC particulier (programme

100000 et port 111. Son rôle est d'associer le numéro de port de la socket d'écoute à un numéro de service RPC donné. Il faut bien sur que `portmap` soit actif sur la machine du serveur pour que ce mécanisme fonctionne.

2.4 Consultation des services existants

2.4.1 Dans un code C

La liste des services déjà enregistrés sur une machine est dans le fichier `/etc/rpc`. Les informations relatives à un service sont :

- son nom officiel
- son numéro
- une liste d'alias

La consultation de ce fichier peut être effectuée par les fonctions :

```
struct rpcent * getrpcbyname(const char *nom_service);
struct rpcent *getrpcbynumber(const int numero_service);
```

qui renvoient la structure

```
struct rpcent
{
  char     *r_name;         /* Nom officel du programme rpc */
  char     **r_aliases;     /* Liste des alias              */
  int      r_number;        /* Numero rpc du programme      */
};
```

2.4.2 En ligne

La commande `rpcinfo` permet d'obtenir des informations sur la liste des services ou un service particulier d'une machine donnée.

1. `rpcinfo -p [ machine ]` : liste des services enregistrés sur une machine
2. `rpcinfo -u machine programme [ version ]` : appel du programme UDP sur la machine donnée. Le programme peut être désigné par son numéro ou un nom qui lui est associé dans le fichier `/etc/rpc`.
3. `rpcinfo -t machine programme [ version ]` : idem pour un service en mode TCP.

2.5 Les différents niveaux d'utilisation

RPC offre 3 couches de fonctions au programmeur. La première couche demande une connaissance quasiment nulle du protocole, la dernière manipule des mécanismes de bas niveau tels que les sockets.

La couche haute : elle proposent des fonctions qui ne peuvent qu'appeler des programmes déjà existants. Cette couche cache le maximum de détails à l'utilisateur mais ne permet pas de développer de nouveaux services.

La couche intermédiaire : ses fonctions demandent un minimum de connaissance sur les protocoles XDR et RPC. Cette couche permet de développer de nouveaux services.

La couche basse : on l'utilise quand les valeurs par défaut des fonctions de la couche intermédiaire ne sont pas adaptées au service que l'on veut développer.

Pour une introduction au protocole RPC, la couche intermédiaire est la plus intéressante. Nous nous pencherons donc uniquement sur celle-ci.

3 La couche intermédiaire

Fichier à inclure :

```
#include <rpc/types.h>
#include <rpc/xdr.h>
```

Cette couche implémente le protocole RPC en s'appuyant sur le protocole UDP. La taille des messages qui transitent est donc limitée (fixée à `UDPMSGSIZE`).

3.1 Côté serveur

Voici les étapes dans la création d'un serveur :

3.1.1 L'écriture des fonctions

Dans le mécanisme RPC, une fonction doit regrouper tous ses paramètres d'entrée dans une structure afin de les décoder par une fonction XDR. Donc, l'unique paramètre d'entrée de la fonction est un pointeur sur cette structure. De même, tous les résultats sont affectés dans une même structure. Selon la fonction, celle-ci peut être allouée statiquement ou dynamiquement. Le seul paramètre de sortie de cette fonction est donc un pointeur du type "générique" `char *` sur cette structure.

3.1.2 L'enregistrement du service

Il consiste à enregistrer individuellement chacune des procédures qui compose le service auprès du programme particulier `portmap`. Cet enregistrement s'effectue par la commande :

```
int registerrpc( rpcprog_t no_prog,
                 rpcvers_t no_version,
                 rpcproc_t no_proc,
                 xdrproc_t xdr_param,
                 xdrproc_t xdr_result);
```

Paramètres :

1. no_pg : numéro du programme où enregistrer la fonction
2. no_vers : numéro de version
3. no_proc : numéro de procédure à donner à la fonction
4. fonction : pointeur sur la fonction à enregistrer
5. xdr_param : fonction XDR d'encodage/décodage des paramètres
6. xdr_result : fonction XDR d'encodage/décodage du résultat

Retour de la fonction :

- En cas de succès → 0
- En cas d'erreur → -1 + envoi d'un message d'erreur sur `stderr`

Au premier enregistrement d'une fonction, le numéro de programme est réservé sur la machine du serveur (celle où s'exécute `portmap`) et la fonction 0 est créée automatiquement (par convention, la procédure 0 ne prendra pas de paramètres et ne renvoie rien, elle sert juste à tester si un numéro de programme particulier existe).

Pour l'instant, le service n'est pas encore disponible. Il faut pour cela créer le processus de service qui l'activera.

3.1.3 Activation du service

La solution la plus simple est de faire jouer le rôle du processus de service par celui qui a créé l'enregistrement de ce service. Dans ce programme, après l'enregistrement, il suffit de faire appel à la fonction :

```
void svc_run(void);
```

Cette fonction réalise une attente en lecture sur la socket du processus.

3.2 Côté client

3.2.1 Appel d'une procédure distante

Un appel de procédure distante se réalise par la fonction :

```
callrpc( char * nom_machine,
         rpcprog_t no_prog,
         rpcvers_t no_version,
         rpcproc_t no_proc,
```

```
        xdrproc_t xdr_param,
        char * param,
        xdrproc_t outproc,
        char *out);
```

Paramètres :

1. nom_machine : nom de la machine où se trouve la fonction à exécuter
2. no_prog : numéro du programme à appeler
3. no_version : numéro de la version du programme
4. no_proc : numéro de la procédure dans le programme
5. xdr_param : fonction d'encodage/décodage XDR pour les paramètres
6. param : pointeur sur la structure des paramètres
7. xdr_result : fonction d'encodage/décodage XDR pour le résultat
8. result : pointeur sur la structure du résultat
   (a) *Avant l'appel* : la place mémoire pour la structure doit être réservée
   (b) *Après l'appel* : la structure est affectée

Retour de la fonction :

- En cas de succès $\rightarrow 0$
- En cas d'erreur $\rightarrow \neq 0$

on peut récupérer des explications sur les erreurs de `callrpc` en donnant son résultat à la fonction :

```
void clnt_perrno(const enum clnt_stat stat);
```

cette fonction envoie un message d'erreur sur `stderr`.

3.2.2 Gestion du *timeout*

Cette fonction gère un renvoi automatique des paquets UDP toutes les 5 secondes en cas de non réponse de la procédure distante et le *timeout* est fixé à 25 secondes. Donc un seul appel à `callrpc` peut entraîner au plus 5 appels et donc 5 exécutions de la procédure distante.

Cas où cela peut se produire :

1. L'exécution de la procédure distante est très longue (supérieure à 5 secondes) : le client n'a rien reçu au bout de 5 secondes, donc il ré-émet le même paquet.
2. L'exécution de la procédure est conséquente et elle reçoît plusieurs demandes en même temps.
   Admettons que la fonction distante est en train d'exécuter le service pour un client $C1$. Puis arrive la demande d'un client $C2$, puis celle de

$C3$. $C2$ attendra que le service pour $C1$ soit fini et $C3$ passera après $C1$ et $C2$: au moins $C3$ risque de ne pas recevoir sa réponse à temps (avant 5 sec.) et ré-enverra un paquet.
  - du côté du client : tout se passe bien puisse qu'il recevra une et une seule réponse
  - du côté du serveur : son traitement n'est pas optimisé car il fera des traitements inutiles.

3.3 Conclusion sur la couche intermédiaire

Son faible paramétrage (*timeout* de l'appel de fonction = 25 sec) et son ensemble réduit des fonctions de manipulation (svc_run, registerrpc, callrpc) rendent son utilisation facile. Mais si les valeurs par défaut de cette couche ne conviennent pas il faut utiliser la couche basse. Par exemple, cela peut se justifier si :

- l'on veut modifier le *timeout*
- la taille des paramètres est supérieure à `UDPMSGSIZE` (8800 octets dans notre système)
- on préfère privilégier la sécurité du transfert des informations plutôt que la rapidité (donc TCP)
- ...

4 La couche basse

Si les options adoptées dans la couche intermédiaire ne sont pas adaptées au service, alors on utilise la couche basse.
On est alors presque au même niveau d'abstraction que les primitives de gestion des sockets. Il faut donc créer un point de communication rattaché à une adresse et préciser le protocole de la couche transport que l'on va utiliser.

4.1 Côté serveur

Fichier à inclure :

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

Avec le protocole RPC, les informations ci-dessus, pour un serveur, sont centralisées dans une structure de type `SVCXPRT` dans laquelle on trouve entre autre le descripteur de la socket (champ `xp_sock`) et le numéro de port (champ `xp_port`).

4.1.1 Création d'un service

Il faut d'abord créer un point de communication (une socket) avec la primitive `socket`. Puis pour un

Service UDP :

```
SVCXPRT *svcudp_create(int socket); /* Descripteur socket */
```

Service TCP :

```
SVCXPRT *svctcp_create(int     socket,       /* Descripteur socket      */
                       uint_t  taille_send,  /* Taille buffer emission  */
                       uint_t  taille_recv); /* Taille buffer reception */
```

4.1.2 Enregistrement d'un service

La fonction ci-dessous réalise l'enregistrement **complet** du service ($\neq$ procédure par procédure) et associe le programme à un numéro de port.

```
bool_t svc_register(SVCXPRT    *xprt,      /* Point de com. */
                    rpcprog_t  prognum,    /* No Programme  */
                    rpcvers_t  versnum,    /* No Version    */
                    void (*service)(),     /* Programme     */
                    int        protocol); /* IPPROTO_UDP, IPPROTO_TCP, 0*/
```

4.1.3 Code du service

Dans le code du service les informations transitent par des flux XDR

Accès aux données :

```
bool_t svc_getargs(const SVCXPRT   *xprt,    /* Point de comm.          */
                   const xdrproc_t  xdr_don, /* Fonction de decodage    */
                   caddr_t          pt_don); /* zone reception donnees */
```

Transmission des résultats :

```
bool_t svc_sendreply(const SVCXPRT   *xprt,    /* Point de comm.        */
                     const xdrproc_t  xdr_res, /* Fonction d'encodage   */
                     const caddr_t    pt_res); /* @ stockage resultats */
```

4.2 Côté client

Fichier à inclure :

```
#include <rpc/clnt.h>
```

Le protocole RPC centralise, pour un client, les informations relatives au point de communication dans une structure `CLIENT`.

4.2.1 Création d'un client

Client UDP :

```
CLIENT *clntudp_create(struct sockaddr_in *addr,     /* @ serveur */
                       rpcprog_t           prognum, /* No programme */
                       rpcvers_t           versnum, /* No version */
                       struct timeval      timeout, /* Timeout    */
                       int                *socket); /* Descrip. socket */
```

La socket est soit créé explicitement, soit automatiquement.

Client TCP :

```
CLIENT *clnttcp_create(struct sockaddr_in *addr,
                       rpcprog_t           prognum,
                       rpcvers_t           versnum,
                       int                *socket,
                       uint_t  taille_send,  /* taille buffer emission  */
                       uint_t  taille_recv); /* taille buffer reception */
```

4.2.2 Appel d'une procédure

```
enum clnt_stat clnt_call(CLIENT *clnt,           /* Pt de comm.            */
                         const rpcproc_t procnum,/* Procedure distante     */
                         const xdrproc_t xdr_don,/* Fonction de decodage   */
                         const caddr_t   pt_don, /* Zone reception donnees */
                         const xdrproc_t xdr_res,/* Fonction d'encodage    */
                         caddr_t         pt_res, /* Zone stockage resultats*/
                         const struct timeval timeout); /* Timeout         */
```

4.2.3 Libération d'un client

```
void clnt_destroy(CLIENT *clnt);
```

4.3 Concepts avancés

Cette couche offre des fonctions qui permettent au super-utilisateur
- de n'activer (de créer un processus) que pour les services effectivement appelés par un ou plusieurs clients
- d'authentifier les requêtes des clients (de vérifier leur identification)
- de traiter les erreurs, les anomalies qui surviennent lors de l'exécution ou de l'appel de fonctions distantes