
EUROGAM Project
Using Sun RPC

Edition 0.1
October 1990

. . .

.

CENBG BORDEAUX, FRANCE : P. Kadionik

Ce document fait le point sur les "Remote Procedure Calls" décrits dans le manuel "Network Programming Guide" de Sun microsystems (pp 1-129). On s'y référera pour y puiser les exemples pratiques indispensables pour une meilleure compréhension de présent document [1]†...

1 Généralités :

1.1 Historique :

La communication entre processus a toujours été un point essentiel sous UNIX. Ainsi, jusqu'à la version 7 d' UNIX, on n'avait que le *pipe* comme seul moyen de communication inter-processus. Mais le *pipe* avait ses limites, car pour communiquer, les processus devaient avoir la même filiation. De plus, l'exécution des processus devait se faire sur la même machine.
Avec la version System III d' ATT, on a introduit les "tubes nommés" (*named pipes*), ce qui a permis la communication entre processus sans filiation. Mais, les processus s'exécutaient toujours sur la même machine...
L'université de Berkeley a développé parallèlement, une autre version d' UNIX (UNIX BSD), mais orientée réseaux. Avec l'arrivée d' UNIX BSD 4.3, le programmeur a à sa disposition des mécanismes puissants : *sockets* (comme interface d'accès au réseau), les protocoles TCP-UDP/IP d'Arpanet, les RPC et XDR.
De son côté, ATT, avec UNIX System V, a développé une interface réseau équivalente aux *sockets*, les *streams*, pour avoir accès aux protocoles Arpanet. Ils ont aussi introduit les mécanismes de *zone mémoire partagée* ainsi que les *sémaphores*.
Mais, actuellement, les mécanismes offerts par UNIX BSD 4.3 se sont imposés comme standard car leur mise en œuvre est aisée.

---

[1] † Le lecteur connaissant déjà les protocoles TCP-UDP/IP et les RPC pourra passer directement au chapître 3.

<table>
<tr><td colspan="2">APPLICATION</td></tr>
<tr><td>Socket</td><td>Stream</td></tr>
<tr><td>TCP UDP</td><td>Protocole 1</td></tr>
<tr><td>IP</td><td>Protocole 2</td></tr>
<tr><td>Arpanet Ethernet</td><td>Réseau 1 Réseau 2</td></tr>
<tr><td>UNIX BSD 4.3</td><td>UNIX System V</td></tr>
</table>

Tableau 1

1.2 L'architecture Arpanet :

Dans les années 70, le DARPA (Defense Advanced Research Project Agency) a développé la technologie Internet pour la transmission par paquets de l'information. En 1980, le DARPA a implanté TCP/IP sur ses machines qui étaient reliées à Arpanet (Arpanet a été développé par le Department of Defense-USA).
L'environnement TCP/IP s'est alors affirmé comme un standard.
L'université de Berkeley a ensuite implanté TCP/IP sous UNIX BSD pour permettre l'accès au réseau Arpanet. L'architecture Arpanet se découpe comme suit :

3	couche application	telnet, ftp, rlogin...
2	couche transport	TCP/UDP
1	couche réseau	IP
0	couche physique	Arpanet, Ethernet...

Tableau 2

On voit que l'on reste tout de même en accord avec les couches basses de la norme ISO.

1.3 Le protocole IP :

Le protocole IP (Internet Protocole) permet de connecter des machines sur des réseaux différents (Arpanet, Ethernet...). C'est un service sans connexion (chaque datagramme (paquet) est considéré individuellement) et non fiable car on n'assure pas l'acheminement vers le destinataire.

1.4 Le protocole UDP :

Le protocole UDP (User Datagram Protocole) se situe au niveau de la couche transport et utilise le service IP pour acheminer les messages. C'est un service sans connexion (orienté transaction) et non fiable, mais il est simple à mettre en œuvre. On a introduit à ce niveau la notion de numéro de port (*port number*) qui permet d'identifier au sein d'une machine, un processus qui utilise ce protocole.
UDP est surtout utilisé pour des réseaux fiables (faible taux d'erreur) comme*tftp* (trivial ftp).

1.5 Le protocole TCP :

Le protocole TCP (Transport Control Protocol) est lui orienté connexion et est fiable.

1.6 RPC et XDR :

Les RPC (Remote Procedure Call) mettent en jeu un ensemble de mécanismes qui permettent l'exécution de procédures sur une machine distante, exécution demandée par la machine locale. Les RPC utilisent les protocoles TCP/IP ou UDP/IP. L'interfaçage réseau (*socket*) est complétement transparent pour le programmeurL'utilitaire *rpcgen* de Sun permet de mettre en place rapidement un système Client (local) /Serveur (local ou distant) RPC.

XDR (eXternal Data Representation) permet une représentation des données indépendantes machine. De façon générale, les données "locales" sont codées sous format XDR par sérialisation, transmises à la machine "distante", puis décodées par désérialisation. On s'affranchit ainsi de la contrainte de représentation des données qui reste propre à une machine.
Les RPC utilisent XDR pour la Sérialisation/Désérialisation des informations (données et résultats).

2 Les RPC :

2.1 Le modèle RPC :

Comme l'on vient de le voir, les RPC permettent à un processus d'une machine locale ("local client machine"), d'exécuter une procédure sur la machine locale ou sur une machine distante ("local/remote server machine").
Les RPC sont basées sur le modèle Client/Serveur. On distingue le *processus client* et le *processus serveur* :
1. Le processus client va essayer de rentrer en contact avec le processus serveur en s'adressant au *service portmapper* de la machine distante. Quand le *portmapper* aura identifié le processus serveur demandé, celui-ci renvoie toutes les coordonnées de celui-ci (rõle du *port number*) pour que le processus client puisse alors s'adresser directement au processus serveur.
2. Le processus client lance la procédure d'appel au processus serveur.
3. Quand la requête arrive au processus serveur, celui-ci la sert et exécute la procédure RPC appropriée, puis renvoie les résultats, s'il y en a, au processus client.

- La procédure d'appel doit contenir les informations nécessaires à identifier la procédure RPC que doit exécuter le processus serveur.
- Le processus serveur est en général endormi (*daemon*). Il ne se réveille que pour servir la requête d'un client RPC. Quand le processus serveur est actif, le processus client est en sommeil jusqu'à ce que le processus serveur ait fini de servir la requête du processus client.
- Les données à transmettre au processus serveur sont sous format XDR. Les résultats transmis au processus client sont aussi sous format XDR.
- Le time-out des RPC intervient à ce niveau (voir 4.4). Quand il n'y a pas de réponse du serveur au bout d'un certain temps défini par le programmeur lors d'un appel RPC, le processus client redevient actif...

2.2 Le service portmapper :

Un processus client doit être capable d'entrer en contact avec le processus serveur RPC. Pour cela, il faut qu'il connaisse le numéro de port (*port number*) du serveur de la machine distante.
Le service portmapper d'une machine Sun est un service qui permet de façon standard aux clients de connaître le numéro de port d'un serveur.
La gestion du service portmapper est assurée par le *serveur portmapper* qui est lancé lors du boot de la machine Sun. Un ensemble de serveurs RPC est activé lors de cette phase de boot. On a en particulier le serveur "rusersd" qui est utilisé lorque l'on veut connaître les utilisateurs en session sur tout le réseau local (% rusers). On retrouve aussi le *serveur portmapper* dont

le rôle est de communiquer le numéro de port d'un serveur répertorié dans son service portmapper à tout client qui en fait la demande. Il faut donc, au préalable que le serveur RPC que l'on désire installer soit déclaré au niveau du service portmapper de la machine distante avant tout appel RPC par un client !

2.3 Reconnaissance d'une procédure RPC :

Un serveur RPC peut avoir à gérer plusieurs procédures. Pour être distinguée, une procédure est identifiée par un numéro de procédure. En plus du numéro de port attribué par le service pormapper, le serveur RPC possède un numéro de programme et un numéro de version. Pour s'adresser à un processus serveur RPC, le processus client doit donc connaître
:
- le nom de la machine où est installé le serveur (adresses IP).
- le numéro de port.
- le numéro de programme.
- le numéro de version.
- le numéro de la procédure à exécuter.
- le type du protocole à employer (TCP ou UDP) avant tout appel RPC.

L'ensemble de ces informations est accessible à l'utilisateur lorsqu'il rentre la commande :
% rpcinfo -p *hostname*

2.4 Choix du protocole :

Le choix du protocole (TCP ou UDP) pour le serveur est laissé au programmeur. Néanmoins, UDP ne permet qu'un échange de données de taille inférieure à 8 Ko...

3 Mise en oeuvre des RPC par rpcgen :

3.1 Le but :

rpcgen est un outil (compilateur) qui permet de créer très rapidement des programmes serveur et client RPC. *rpcgen* demande peu de connaissances.

3.2 Le principe :

1. Le programmeur écrit le fichier de la structure de données qui sera utilisée par la procédure RPC (pgr.x) en utilisant le langage RPC (*RPC*

language) qui reste très proche du C.
2. Puis, il lance l'utilitaire rpcgen avec comme paramètre, le fichier précédent. En retour, un squelette du client (pgr_clnt.c), un squelette du serveur (pgr_svc.c), un fichier des données sous format XDR (pgr_xdr.c) et un fichier header contenant les constantes (pgr.h) sont générés.
3. Le programmeur écrit en C sa procédure RPC (proc.c) et un fichier qui va permettre de générer le programme serveur et le programme client (client.c).
4. Le programmeur compile client.c avec pgr_clnt.c et pgr_xdr.c pour obtenir le programme exécutable *client*.
5. Le programmeur compile proc.c avec pgr_svc.c et pgr_xdr.c pour obtenir le programme exécutable *serveur*.
6. Le programme serveur est lancé sur la machine distante (par la commande %*serveur* &) prẽt à servir une requẽte RPC.

3.3 Un exemple d'application :

L'exemple choisi est tiré du manuel Sun cité au début du document. S'y référer pour trouver les sources des fichiers cités...

On désire réaliser un programme client qui imprime à l'écran d'une machine distante un message.

- Etape 1 : on crée le fichier en "langage RPC" *msg.x* qui spécifie la structure de données utilisée par les RPC.

```
program MESSAGEPROG {
    version MESSAGEVERS {
        int PRINTMESSAGE (string) = 1;
    } = 1;
} = 0x20000000;
```

fichier msg.x

Le numéro de procédure vaut 1.
Le numéro de version vaut 1.
Le numéro de programme vaut 0x20000000 (le numéro de programme doit appartenir à une plage donnée).
La structure de données est ici :
- une chaîne de caractères (le message à afficher) utilisée lors de l'appel RPC.
- un entier utilisé lors du retour d'appel RPC.

- Etape 2 : on fait appel à rpcgen :

```
% rpcgen msg.x
```

On récupère alors les fichiers *msg_clnt.c*, *msg_svc.c*, et *msg.h*. Notons qu'ici, on n'a pas de fichier *msg_xdr.c* car la structure de données employée est simple.

- Etape 3 : il faut ensuite écrire la procédure RPC *msg_proc.c* et le programme C *rprintmsg.c* qui va permettre de "lier" le tout.

- Etape 4 : on génère le programme client :

```
% cc rprintmsg.c msg_clnt.c -o msg_client
```

- Etape 5 : on génère le programme serveur :

```
% cc msg_proc.c msg_svc.c -o msg_server
```

- Etape 6 : on lance le programme serveur sur la machine distante% rlogin *remote_hostname*

```
% msg_server &
```

Il ne reste plus qu'à utiliser le programme client si l'on désire imprimer un message à l'écran de la machine distante...

3.4 Quelques remarques :

- Il faudra rajouter lors de la génération des programmes client et serveur le fichier *pgr_xdr.c* si nécessaire.
- Pour la procédure RPC, on n'utilise que des pointeurs sur les données et sur les résultats.
- Le nom de la procédure est en majuscules suivi du numéro de procédure (ex : PRINTMESSAGE_1, le numéro de version étant ici 1).
- Le langage RPC fait appel à XDR. Pour plus d'informations sur XDR, voir plus loin...

4 Les RPC : programmation avancée :

4.1 Présentation :

Nous venons de voir que *rpcgen* permettait de mettre en place des programmes serveur et client de façon systématique. Remarquons que l'on n'avait aucun moyen d'agir sur les modèles du squelette du serveur et du client générés par *rpcgen*. Nous allons montrer qu'il est possible de se bâtir ses propres programmes serveur et client.
On distingue trois niveaux dans la programmation des RPC :

- *la couche haute* : elle est totalement transparente à l'utilisateur.
- *la couche moyenne* : elle permet de construire des applications de couche haute.
- *la couche basse* : elle permet de choisir le protocole de transport, le temps de time-out des RPC...

4.2 La couche haute de RPC :

C'est en fait un ensemble de routines offertes à l'utilisateur. L'aspect RPC lui est totalement masqué.
Sur Sun, les services offerts sont divers :
- rusers() : permet d'avoir des informations sur les utilisateurs en session sur le réseau.
- havedisk() : permet de savoir si une machine du réseau possède un disque dur.
- yppasswd() : met à jour le mot de passe d'un utilisateur si les Yellow Pages sont utilisées pour l'administration du réseau.
- ...

Ces routines sont archivées sous /usr/lib/librpcsvc.a .

4.3 La couche moyenne de RPC :

Dans cette couche, on a à écrire le programme serveur et le programme client. Cette couche permet de créer des services de la couche de haut niveau.

Reprenons l'exemple la routine rusers() (la commande Sun % rusers utilise la routine rusers()). En tapant la commande :
% rpcinfo -p *hostname*,
on a vu qu'il existait un serveur s'appelant ruserd.
En fait, le serveur de couche moyenne est ruserd. Pour construire le programme client (seule chose qu'il y aurait à faire mais qui est déjà réalisée), on utilise les informations se trouvant sous /usr/include/rpcsvc/ . Le résultat de tout cela est bien sûr la routine de couche haute rusers()
Si l'on veut maintenant créer notre propre serveur et notre propre client, on a à disposition 3 outils de base :
- **callrpc()** : cette routine permet, côté client, de faire des appels RPC au serveur.
- **registerrpc()** : cette routine permet, côté serveur, d'initialiser le serveur. En fait, on déclare le serveur au service portmapper (affectation d'un numéro de port) et on crée l'interfaçage réseau (socket).
- **svc_run()** : cette routine permet, côté serveur, d'attendre un appel

RPC, transmet la demande à la procédure RPC, renvoie les résultats au client s'il y en a, puis recommence à attendre un appel RPC.

Remarques :

On doit préciser dans l'appel callrpc() les structures XDR (filtres) à utiliser pour l'acheminement des données et des résultats. On doit aussi donner en paramètres un pointeur sur la structure de données utilisée pour les données (pour l'initialisation de l'appel RPC) et un pointeur sur la structure de données utilisée pour les résultats (pour le traitement de ceux-ci après le retour d'appel).
On a juste à écrire la procédure RPC et les structures XDR pour les données et les résultats.
On peut se servir de *rpcgen* pour obtenir les structures XDR (fichier pgr_xdr.c).
On n'a pas le choix du protocole, qui est ici UDP/IP.
On trouvera des exemples dans le manuel Sun.

4.4 La couche basse de RPC :

C'est à ce niveau que l'on peut contrõler les RPC.
On peut choisir le protocole à employer (TCP/IP ou UDP/IP). TCP/IP permet d'avoir des échanges de données supérieure à 8 Ko.
On peut changer le time-out des appels RPC (fixé par défaut à 30 secondes).
On peut installer un système de reconnaissance des appels (qui est le client ?) et donc permettre un aiguillage au niveau des procédures RPC suivant le client.
rpcgen génère des squelettes appartenant à la basse couche de RPC.
On trouvera là aussi des exemples dans le manuel Sun.

4.5 Autres types de RPC :

4.5.1 Les "Raw RPC" :

Ce type de RPC n'est utilisé que pour la phase de développement d'un système Client/Serveur RPC. Par rapport à la couche basse de RPC, on n'enregistre pas le serveur au niveau du service portmapper. On n'a pas accès aux paramètres de contrõle des appels RPC.

4.5.2 les "Broadcast RPC" :

Cela permet, entre autre, de rechercher des informations en s'adressant à **tous** les serveurs des machines connectées sur le réseau.
Pour s'adresser juste à un ensemble de serveurs, il faut donc créer une interface supplémentaire ("serveur de serveurs").
Les "broadcast RPC" n'utilisent qu' UDP/IP.

5 La représentation de données XDR :

5.1 Rappels :

XDR (eXternal Data Representation) est un ensemble de routines qui permet de représenter des structures de données ne dépendant pas de la représentation interne de celles-ci d'une machine. XDR permet donc l'échange d'informations entre machines hétérogènes. Pour aller plus loin, on peut dire que XDR règle les problèmes de portabilité de données et peut être utilisée dans n'importe quel programmeLa représentation XDR est utilisée par les RPC.

Pour utiliser XDR, il faut inclure dans les fichiers :
#include <rpc/rpc.h>

5.2 La bibliothèque XDR :

Les routines XDR sont de la forme :

```
xdr_xxx(xdrs, xp)
    XDR *xdrs ;
    xxx *xp ;
{
...
}
```

Voici un exemple de structure XDR pour un entier long :

```
xdr_long(xdrs, xp)
    XDR *xdrs ;
    long *xp ;
```

On utilise la même structure aussi bien pour la sérialisation que pour la désérialisation.
On utilise toujours, comme on l'a déjà vu des pointeurs sur les objets et

non les objets eux-même.
On trouvera de nombreux exemples dans le manuel Sun...

6 Conclusion :

La mise en place de RPC Sun est relativement simple mais cela nécessite une bonne compréhension des mécanismes de base décrits dans ce document. Il est tout aussi important de "se faire la main" en mettant en place de petites applications. Le manuel "Networks Programming Guide" de Sun vous sera pour cela très utile...
Signalons enfin que les RPC Sun sont reconnus de tous. Wind River les a implantés dans son produit Temps Réel VxWorks (compatibilité complète). On peut maintenant installer un serveur RPC Temps Réel sous VxWorks alors que le client RPC est installé sous Sun (non Temps Réel). Un nouvel horizon s'est ainsi ouvert pour le programmeur...