Exécution à distance

(/home/terre/d01/adp/bcousin/Polys/RPC.fm- 27 Décembre 1999 10:50)

Plan

- Introduction
- RPC
- Les messages RPC
- La programmation des RPC
- XDR
- Conclusion

Bibliographie

- J-M. Rifflet, La communication sous Unix, EdiScience, 1996.
- M. Gabassi, B. Dupouy, L'informatique répartie sous Unix, 1992.
- ONC+ Developper's Guide, Solaris Software Developper Collection, Sun, 1999.

1. Introduction

Besoin d'un environnement de haut niveau pour le développement d'applications réparties qui :

- reprend le concept du client/serveur
- permet d'identifier un très grand nombre de services (> #n° de port)
- conserve les paradigmes habituels d'exécution :
  - l'appel de fonction, passage de paramètres
  - la notion de programme (ensemble de fonctions)

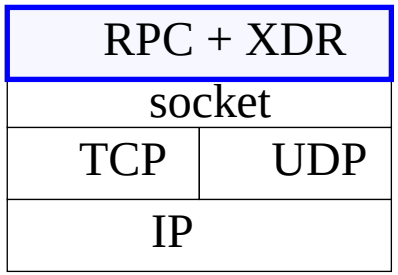
    ☞ RPC
- masque l'hétérogénéité de représentation des données
  - format standard, fonctions de transcodage

    ☞ XDR
  - description des structures de données
    . langage de description : RPC language
  - accompagné d'un outil de génération automatique :

    ☞ RPCgen
- tout en offrant des services supplémentaires :
  - d'authentification, de "broadcast", de "batching", de "call back", etc.

Un environnement de plus haut niveau que les “sockets” et la transmission de messages.

⇨ Architecture fonctionnelle sur Internet :



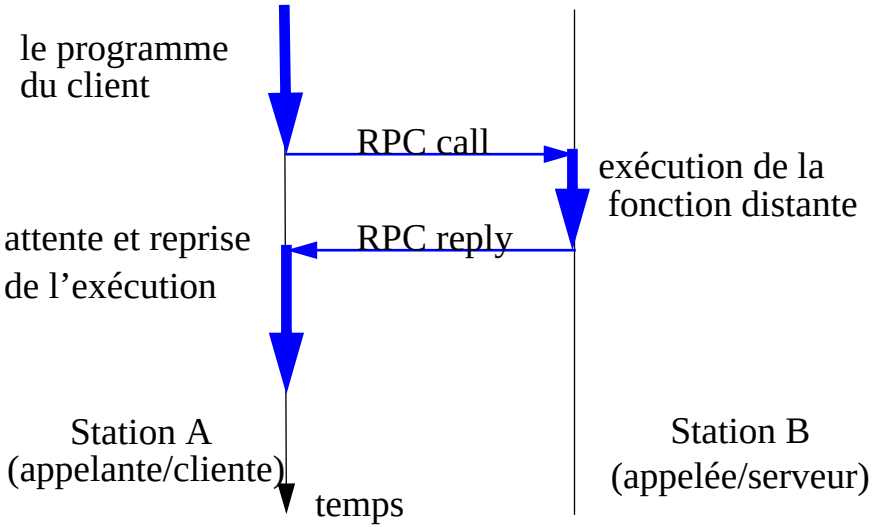
- TI-RPC = Transport independent RPC
  - Implémentation des RPC qui permet le développement d’applications indépendamment des éléments logiques et physiques (réseaux, protocoles, etc.) utilisés pour transmettre des données.

2. Le RPC

2.1. Présentation

❑ Remote Procedure Call (rfc 1831) version 2

Exécution d'une procédure à distance :



2.2. La sémantique de l'appel :

- Si on tient compte des erreurs (pertes, duplications) pouvant survenir lors des communications, on définit 3 sémantiques possibles pour l'appel de procédures distantes :
  - exactement une fois
  - au moins une fois
  - au plus une fois.
- La sémantique choisie par l'implémentation sous RPC-Sun est *au moins une fois*
  - de ce fait il faut s'assurer que l'exécution d'une procédure distante soit idempotent,
    - par exemple en utilisant le numéro de transaction (xid) disponible dans chaque message RPC.

2.3. Les paramètres

- un seul paramètre est échangé lors de l'appel (RPC call)
  - si l'application requiert l'échange de plusieurs paramètres ils doivent être regroupés au sein d'une seule structure de données.
- un seul élément peut être échangé lors du retour (RPC reply)
  - à travers la valeur de retour de la fonction

2.4. Identification des procédures distantes

- Une procédure distante est identifiée de manière unique par un triplet :
  - #program, #prog_version, #procedure
- Un programme regroupe un ensemble de procédures et possède une version
  - plusieurs versions peuvent être disponibles simultanément
- Certains numéros de programmes sont réservés à certains services :

Tableau 1 : les numéros de programme

numéro de programme	utilisation
0000.0000 - $1FFF.FFFF_{16}$	pour des services généraux
2000.0000-$3FFF.FFFF_{16}$	pour des services en cours de développement
4000.0000-$5FFF.FFFF_{16}$	attribués dynamiquement
6000.0000-$FFFF.FFFF_{16}$	réservés

- Fichier */etc/rpc*
  - L'association entre service RPC et numéro de programme est décrit par le fichier */etc/rpc* :

```
rpcbind         100000  portmap sunrpc rpcbind
rstatd          100001  rstat rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
etherstatd      100010  etherstat
rquotad         100011  rquotaprog quota rquota
sprayd          100012  spray
...
```

❑ Commande *rpcinfo*

Liste les programmes, versions et procédures disponibles sur une station :

```
$ rpcinfo
     program version netid     address                service    owner
    100000    4    ticots    pondichery.rpc      portmapper superuser
    100000    3    ticots    pondichery.rpc      portmapper superuser
    100000    4    ticotsord pondichery.rpc      portmapper superuser
    100000    3    ticotsord pondichery.rpc      portmapper superuser
    100000    4    ticlts    pondichery.rpc      portmapper superuser
    100000    3    ticlts    pondichery.rpc      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
    100007    3    udp       0.0.0.0.128.5       ypbind     superuser
    100007    2    udp       0.0.0.0.128.5       ypbind     superuser
    100007    1    udp       0.0.0.0.128.5       ypbind     superuser
    100007    3    tcp       0.0.0.0.128.3       ypbind     superuser
    100007    2    tcp       0.0.0.0.128.3       ypbind     superuser
    100007    1    tcp       0.0.0.0.128.3       ypbind     superuser
     ...
```

Nota : on remarque les différents services protocolaires, numéros de port, formes d'adresse

2.5. Sélection du protocole chargé des communications

☞ on a le choix entre plusieurs protocoles : TCP, UDP, etc.

- lors de la création de client : clnt_create()
  - explicitement : “udp”
  - par la variable d’environnement : $NETPATH
  - par le fichier */etc /netconfig* :

```
# The "Network Configuration" File.
# Each entry is of the form:
#       <network_id> <semantics> <flags> <protofamily> <protoname> \
#               <device> <nametoaddr_libs>
# [...] (v = visible)
udp        tpi_clts      v     inet    udp    /dev/udp        -
tcp        tpi_cots_ord  v     inet    tcp    /dev/tcp        -
rawip      tpi_raw       -     inet    -      /dev/rawip      -
ticlts     tpi_clts      v    loopback -      /dev/ticlts     straddr.so
ticotsord  tpi_cots_ord  v    loopback -      /dev/ticotsord  straddr.so
ticots     tpi_cots      v    loopback -      /dev/ticots     straddr.so
```

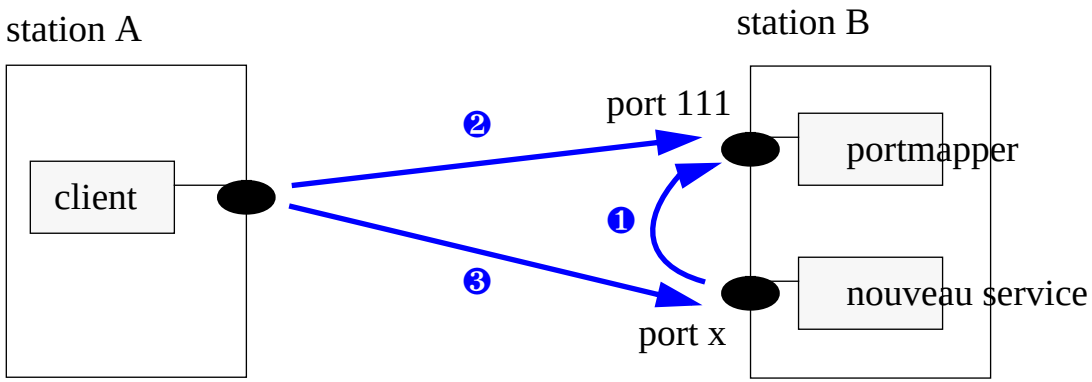
- lors du lancement du serveur :
  - choix simple ou multiple

2.6. L'huissier :"portmapper or rpcbind process"

L'huissier permet de rediriger un client vers le numéro de port hébergeant le service

L'huissier est sur un numéro de port réservé : 111

- Les clients n'ont besoin de connaître que ce seul numéro de port/service



Les fonctions de l'interface de programmation :

- pmap_set() = 1 ; enregistre un service ❶
- pmap_unset() = 2 : désabonne un service
- pmap_getport() = 3 : retourne le numéro de port associé au service ❷
- pmap_getmaps() = 4 : liste les services présents
- pmap_rmtcall() = 5 : appel d'une procédure distante ❸

- le Portmapper est lui-même un service RPC :
  - description des procédures *portmap* en langage RPC
  - description de la structure des messages *portmap* en langage XDR
  - cf. le service *rpcbind* déclaré dans *etc/rpc*

2.7. XDR/RPC language

❏ Langage XDR

Langage de description de la structure des données échangées lors du RPC

- Défini par le rfc 1014.

Syntaxe issue de celle employée pour la description des données dans le langage C

- Cf exemple donné pour la description des messages RPC

❏ Langage RPC

Extension du Langage XDR, pour permettre la définition de procédure(programme)

☞ RPC language = XDR language + program_def + version_def + procedure_def !

- Cf exemple donné pour la description du service PING

2.8. Exemple de description de service

Décrit en “RPC language”

```
program PING_PROG {
        /*Latest and greatest version*/
        version PING_VERS_PINGBACK {
           void
           PINGPROC_NULL(void) = 0;
            /* Ping the client, return the round-trip time
            * (in microseconds). Returns -1 if the operation
            * timed out.*/
           int
           PINGPROC_PINGBACK(void) = 1;
        } = 2;

        /* Original version*/
        version PING_VERS_ORIG {
           void
           PINGPROC_NULL(void) = 0;
        } = 1;
} = 100115;

const PING_VERS = 2;      /* latest version */
```

On remarque qu’aucune description de structure de données n’existe :

- il n’y en a aucune ! → void

3. Les messages du protocole RPC

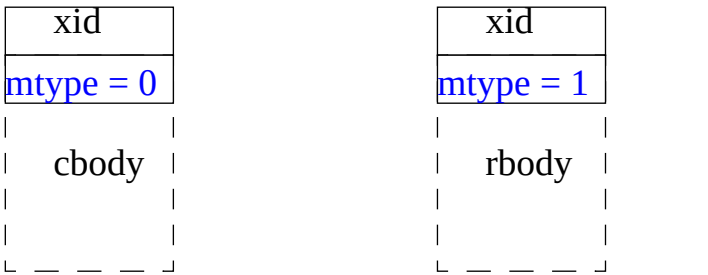
3.1. La structure générale du message

- 2 types de message (en XDR description language !) :

```
enum msg_type {
     CALL = 0,
     REPLY = 1
};
```

- Structure générale d'un message RPC :

```
struct rpc_msg {
         unsigned int xid;
         union switch (msg_type mtype) {
         case CALL:
            call_body cbody;
         case REPLY:
            reply_body rbody;
         } body;
      };
```

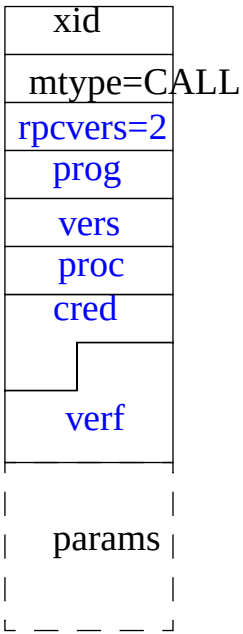


Call RPC message

Reply RPC message

3.2. La structure d'un message "RPC call"

```
struct call_body {
        unsigned int rpcvers;       /* must be equal to two (2) */
        unsigned int prog;
        unsigned int vers;
        unsigned int proc;
        opaque_auth  cred;
        opaque_auth  verf;
        opaque params[0];/* procedure specific parameters start here */
};
```



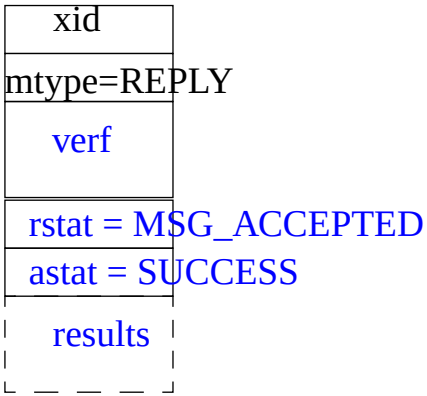
Call RPC message

3.3. La structure d'un message "RPC reply"

```
union reply_body switch (reply_stat rstat) {
      case MSG_ACCEPTED:
         accepted_reply areply;
      case MSG_DENIED:
         rejected_reply rreply;
} reply;

struct accepted_reply {
         opaque_auth verf;
         union switch (accept_stat astat) {
          case SUCCESS:
            opaque results[0]; /*procedure-specific results start here*/
          case PROG_MISMATCH:
             struct {
                unsigned int low; /* lowest RPC program versions */
                unsigned int high;
             } mismatch_info;
          default:
             void;/*Cases incl. PROG_UNAVAIL, PROC_UNAVAIL,GARBAGE_ARGS, and SYSTEM_ERR.*/
          } reply_data;
};

union rejected_reply switch (reject_stat stat) {
      case RPC_MISMATCH:
         struct {
            unsigned int low;
            unsigned int high;
         } mismatch_info;
      case AUTH_ERROR:
         auth_stat stat;
};
```



successful Reply RPC message

4. La programmation des RPC

4.1. Introduction

2 outils complémentaires :

- ❏ Utilisation de l'interface de programmation des RPC :
  - bibliothèque de fonctions
  - 3 ensembles de fonctions définissant 3 niveaux de programmation :
    - élevé, intermédiaire ou bas
- ❏ Un outil de génération automatique de code d'applications réparties utilisant les RPC :
  - RPCGen

4.2. Fonctions du niveau élevé

- getrpcport() : port associé à la version d'un programme sur une machine donnée
- rusers() : les utilisateurs connectés sur une machine
- rnusers() : le nombre d'utilisateurs connectés sur une machine
- rwall() : envoi un message à tous les utilisateurs d'une machine

4.3. Fonctions de niveau intermédiaire

N'utilisent que le protocole UDP, de manière rigide :

- temporisateur de 5s, répétition systématique des messages perdus et au maximum 5 fois, taille maximum des segments 8000 octets, etc

- registerrpc() : enregistrement d'un service (cf. pmap_unset())
- svcrun() : attente des clients par le serveur
- callrpc() : appel de la procédure à distance par le client

4.4. Fonctions de bas niveau pour le serveur

- svcudp_create() : initialisation de la communication (création d'une socket)
- svctcp_create() : préparation de la connexion (création d'une socket)
- svc_register() : enregistrement d'un service, lui associe une fonction de traitement
- svc_getargs() : décodage des arguments de la procédure
- svc_getcaller() : origine de la requête
- svc_freeargs() : libération de l'espace alloué par XDR
- svc_sendreply() : envoi de la réponse
- svc_destroy() : destruction d'une communication (connexion)

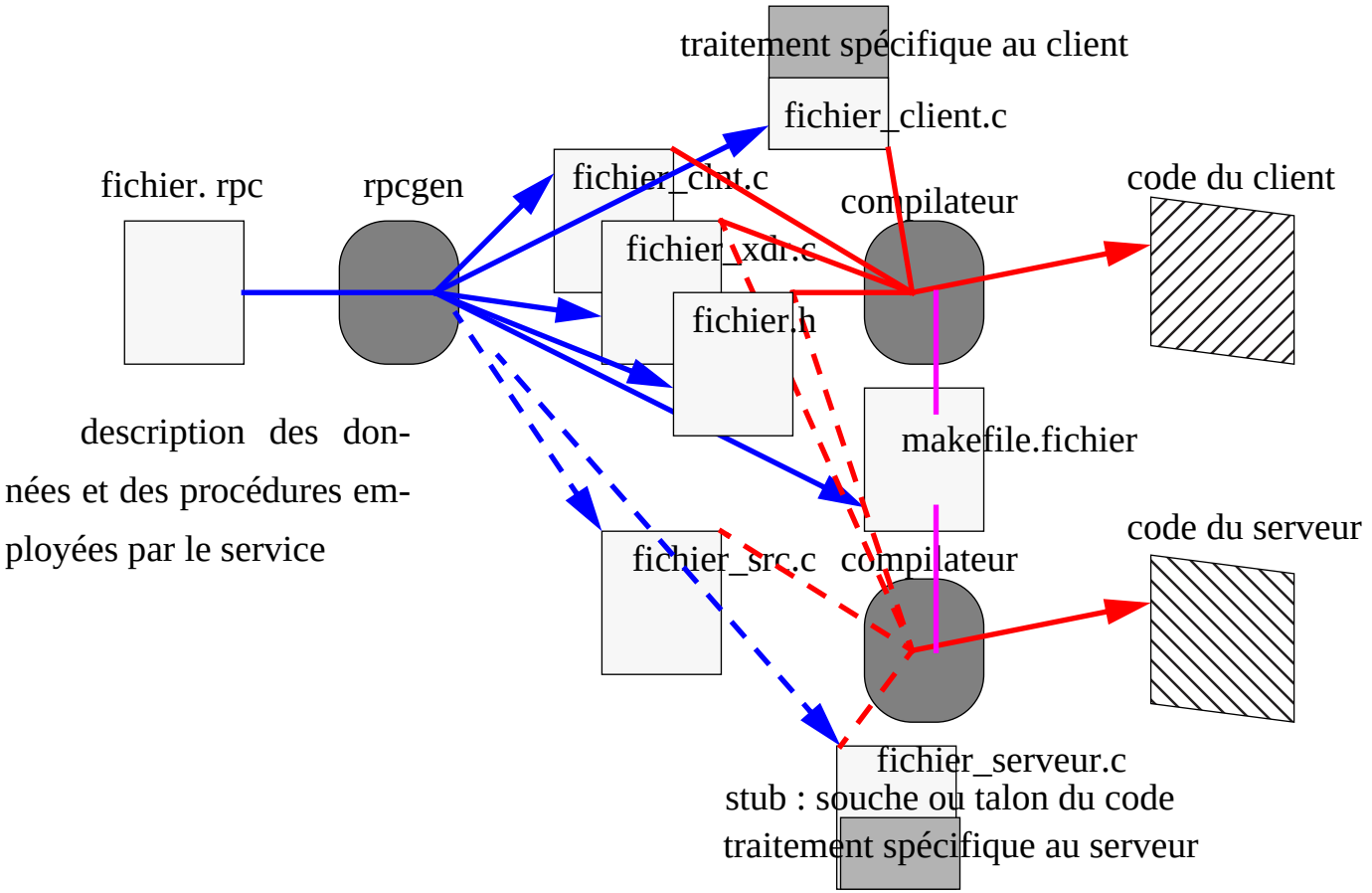
4.5. Fonctions de bas niveau pour le client

- clnt_create() : mise en relation ou établissement de la connexion (deux variantes : clntudp_create(), clnttcp_create())
- clnt_call () : appel de procédure à distance
- clnt_control() : opérations de contrôle
- clnt_freeres() : libération des ressources utilisées par XDR
- clnt_destroy() : libération de la connexion
- clnt_perror() : information sur les erreurs (ou encore clnt_serreor(), clnt_geterr(), svcerr_systemerr())
- authunix_create_default() : fonction d'authentification

4.6. RPCGen

Utilisation de la description des structures de données (XDR language) et de la définition des services (RPC language) pour générer automatiquement le code des appels aux fonctions de codage :

☞ $ rpcgen -a fichier.rpc



5. XDR

5.1. Introduction

XDR : External data representation (RFC 1832)

Définit :

- une technique standard d'encodage pour chaque type de données

Associée à XDR language

- un langage de description des données

5.2. Les principaux types de données

Les types habituels des langages de programmation (présents en langage C plus quelques ajouts) :

- types simples :
  - les nombres entiers (signés, non-signés et hyper), les booléens, les nombres flottants (courts ou longs), les énumérations.
  - et rien (void)
- types structurés :
  - les chaînes de caractères, les tableaux fixes ou variables, les structures opaques fixes ou variables, les enregistrements avec ou sans discriminant.
- les constantes et redéfinition de nom de type (typedef) !

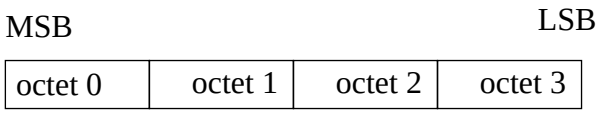
5.3. Encodage de quelques structures de données

5.3.1 Signed Integer

Code les valeurs [-2^{31}, $2^{31}-1$] en complément à 2

Syntaxe : int *identifier*;

Encodage :

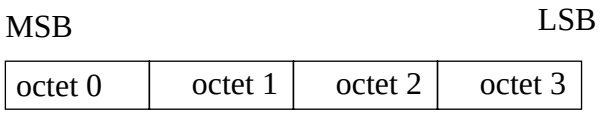


5.3.2 Unsigned Integer

Code les valeurs [0, 2^{32}] en complément à 2

Syntaxe : unsigned int *identifier*;

Encodage :

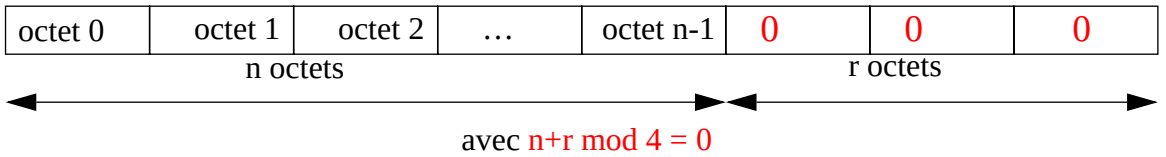


5.3.3 Fixed-length opaque data

Transmet des données (correspondant à *n* octets) qui ne doivent pas être interprétées

Syntaxe : opaque *identifier*[n];

Encodage :

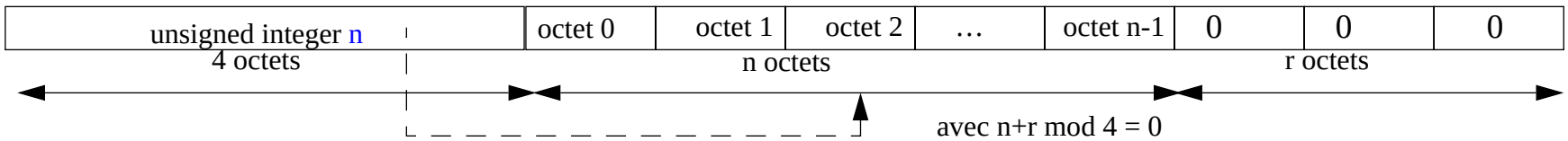


5.3.4 Variable-length opaque data

Transmet des données de longueur variables qui ne doivent pas être interprétées

Syntaxe : opaque *identifier*<n>;

Encodage :

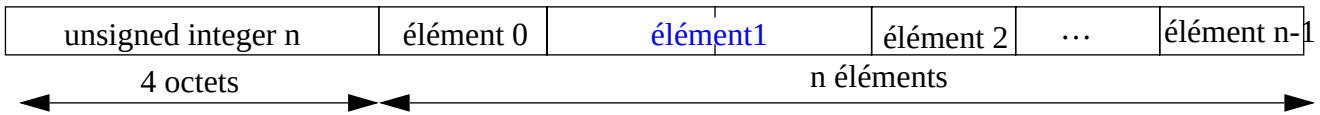


5.3.5 Variable-length array

Transmet un tableau d'éléments de même type en nombre variable. La longueur de chaque élément est multiple de 4 octets, mais ils peuvent avoir des longueurs différentes

Syntaxe : *type identifier*<n>;

Encodage :

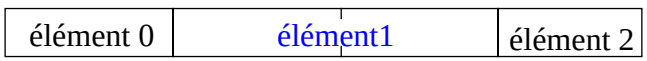


5.3.6 Enregistrement

Transmet un enregistrement de composants. La longueur de chaque composant est multiple de 4 octets.

Syntaxe : structure { *component-declar_1; component_declar_2*} *identifier*;

Encodage :



5.3.7 Une union de composants avec discriminant

Transmet un objet composé d'un discriminant et du composant associé. Le type du discriminant est de type simple (entier, booléen, énuméré).

Syntaxe :

```
union switch ( discriminant ) {
     case discriminant_value_1 : component-declar_1;
     case discriminant_value_2 : component_declar_2;
     default : default_declar;
} identifier;
```

Encodage :

5.4. Fonctions d'encodage

Le système Unix propose un grand nombre de fonctions d'encodage et décodage, dont la manipulation s'avère délicate :

- xdrstdio_create(), xdrmem_create(), xdr_free()
- xdrrec_create(), xdrrec_endofrecord(), xdrrec_skiprecord(), xdrrec_eof()
- xdr_getpos(), xdr_setpos()
- xdr_void(), xdr_char(), xdr_short(), xdr_int(), xdr_long(), xdr_u_short(), xdr_u_int(), xdr_u_long(), xdr_float(), xdr_double()
- xdr_bytes(), xdr_array(), xdr_vector(), xdr_enum(), xdr_hyper()
- xdr_reference(), xdr_pointer(), xdr_complex(), xdr_opaque()
- etc.

L'outil RPCGen génère automatiquement l'appel des bonnes fonctions avec les bons paramètres !

6. Conclusion

L’appel de procédure à distance utilise et prolonge le concept de client/serveur :

- notion de programme, version et procédure distante

On normalise :

- Un protocole d’échange des données : RPC
- Un format standard de transcodage des données : XDR

On propose des langages :

- de description des structures de données : XDR language
- de définition des services distants : RPC language

On utilise des outils :

- génération automatique de code
- fonctions de l’interface de programmation pour RPC et pour XDR

Extensions WEB : WWW client/server, java machine, applet/servlet, cookie, etc.

Le format d’encodage pourrait être variable et déterminé par le concepteur : ASN1 + BER