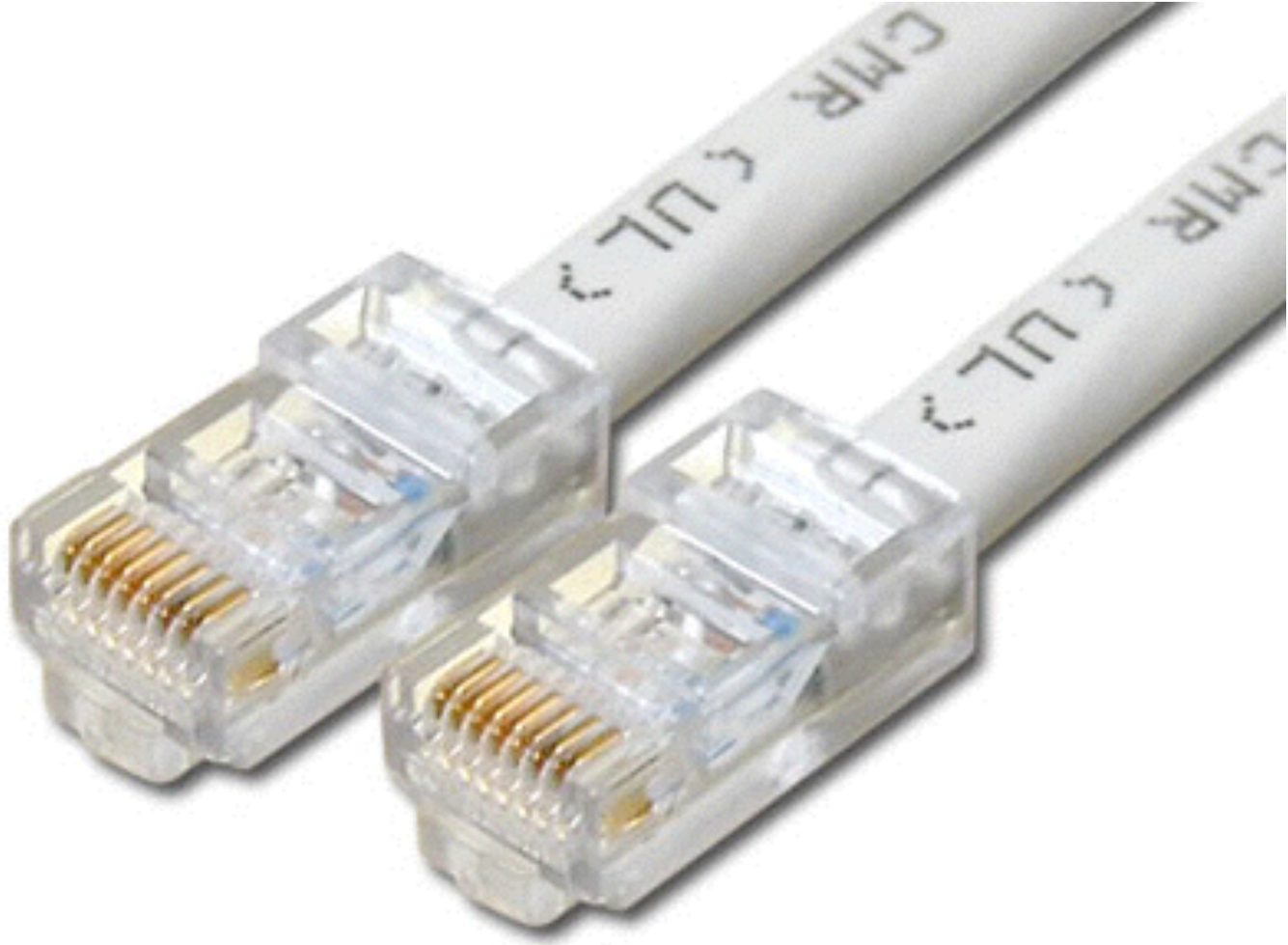
Programmation Réseau

RPC/XDR



Jean-Baptiste.Yunes@univ-paris-diderot.fr

UFR Informatique

2014

Les RPC/XDR du monde Unix

- il s'agit d'un service réseau ancien (ARPANET)
- l'idée est de permettre d'**appeler une fonction distante** (Remote Procedure Call)
  - une telle fonction s'appelle un **service RPC**
- pour se faire, il est nécessaire de **normaliser la représentation des données échangées** paramètre effectifs et retour de fonction, avec XDR (eXternal Data Representation)
- un service très connu NFS (Network File System)

RPC

- la version 2 est normalisée dans la RFC 5531 (Mai 2009)

  - le document original de l'ONC (Open Network Computing) est décrit par la RFC 1831 (Août 1995)

- la première initiative a été normalisée par la RFC 1057 (Juin 1988) à l'initiative de Sun Microsystems®

XDR

- la version la plus récente est décrite dans la RFC 4506 (Mai 2006)

- le document original est la RFC 1014

- Le modèle repose sur l'existence d'un service de nommage
  - il s'agit du service Internet 111 anciennement connu sous le nom de `sunrpc` ou `portmap`
- pour être disponible un service RPC doit s'être enregistré auprès du portmapper
  - le portmapper dispose d'une base des services enregistrés (nom et port TCP ou UDP)

- Un exemple de services disponibles (ici des services tous liés à NFS)

Terminal — ssh — 80×24

```
<liafa0-25-[11:22]-[~]> /usr/sbin/rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  39493  status
    100024    1   tcp  49895  status
    100021    1   udp  36364  nlockmgr
    100021    3   udp  36364  nlockmgr
    100021    4   udp  36364  nlockmgr
    100021    1   tcp  52415  nlockmgr
    100021    3   tcp  52415  nlockmgr
    100021    4   tcp  52415  nlockmgr
<liafa0-26-[11:22]-[~]>
```

- Ici le service supplémentaire est celui des pages jaunes

Terminal — ssh — 74×28

```
<amertume-3-[11:25]-[~]> rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  rpcbind
    100000    3   tcp    111  rpcbind
    100000    2   tcp    111  rpcbind
    100000    4   udp    111  rpcbind
    100000    3   udp    111  rpcbind
    100000    2   udp    111  rpcbind
    100007    3   udp  32780  ypbind
    100007    2   udp  32780  ypbind
    100007    1   udp  32780  ypbind
    100007    3   tcp  32778  ypbind
    100007    2   tcp  32778  ypbind
    100007    1   tcp  32778  ypbind
    100024    1   udp  32786  status
    100024    1   tcp  32781  status
    100133    1   udp  32786
    100133    1   tcp  32781
    100021    1   udp   4045  nlockmgr
    100021    2   udp   4045  nlockmgr
    100021    3   udp   4045  nlockmgr
    100021    4   udp   4045  nlockmgr
    100021    1   tcp   4045  nlockmgr
    100021    2   tcp   4045  nlockmgr
    100021    3   tcp   4045  nlockmgr
    100021    4   tcp   4045  nlockmgr
1073741824    1   tcp  32783
<amertume-4-[11:25]-[~]>
```

- Le fichier des services RPC standard connus est (sous Unix) /etc/rpc :

Terminal — more — 94×29

```
#
# $FreeBSD: src/etc/rpc,v 1.7 1999/08/27 23:23:44 peter Exp $
# rpc 88/08/01 4.0 RPCSRC; from 1.12   88/02/07 SMI
#
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rstat_svc 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
3270_mapper     100013
rje_mapper      100014
selection_svc   100015  selnsvc
database_svc    100016
rexd            100017  rex
alis            100018
sched           100019
llockmgr        100020
nlockmgr        100021
x25.inr         100022
statmon         100023
status          100024
/etc/rpc
```

RPCL

- au même titre qu'il existe un langage de description des types XDR, il existe un langage description des procédures qui opèrent sur ces types
  - ce langage est **RPCL** (Remote Procedure Call Language)
- il est relativement simple et permet de :
  - décrire des types de données
  - décrire un service (avec version)
  - typer les procédures d'une version de service

- les données qui peuvent être décrites en RPCL sont :
  - des énumérations
  - des constantes
  - des définitions d’alias de type
  - des structures
  - des unions

- un exemple d'énumération RPCL :

```
enum {
   LUNDI = 0,
   MARDI = 1,
   MERCREDI = 2,
   JEUDI = 3,
   VENDREDI = 4,
   SAMEDI = 5,
   DIMANCHE = 6
};
```

- un exemple de constante RPCL :

```
const JOURS_PAR_SEMAINE = 7;
```

- un exemple de définition d'alias de type RPCL :

```
typedef int valeurs[255];
```

- les déclarations de variables RPCL peuvent prendre la forme :

  - simple :
    ```
    int valeur;
    ```

  - de tableau de taille fixe :
    ```
    int valeur[255];
    ```

  - de tableau de taille variable :
    ```
    int valeur<255>;
    ```

  - de pointeur :
    ```
    int *pValeur;
    ```

- un exemple de structure RPCL :

```
struct maStructure {
  int valeur;
  int autresValeurs[200];
};
```

- un exemple d'union RPCL :

```
union myUnion switch(int valeur) {
  case 0:
    int iValeur;
  case 1:
    float fValeur;
  default:
    void;
};
```

- un service RPC est appelé `program` en RPCL et est identifié par un numéro de service RPC (auquel correspondra un ou des ports) :

```
program MYSERVICE {

   définition d'une version de service

} = numéro;
```

- la définition de la version d'un service a la forme :

```
version VERSION1_1 {

    déclaration de prototypes

} = numéro;
```

- le numéro identifie le numéro de version de l'API du service (1, 2, 3, etc.)

- la déclaration d’un prototype a la forme de la déclaration de signature d’une fonction C suivi par l’affectation d’un numéro de fonction:

  void UNEFONCTION(int) = 1;

  unsigned int UNEAUTREFONCTION(void) = 2;

- un service de calcul pourrait donc avoir la forme suivante :

```
program PROG {
  version V1 {
    int addition(int,int) = 1;
    int soustraction(int,int) = 2;
  } = 1;
} = 0xDEADBABE;
```

- RPCL utilise certains types particuliers
  - `bool` est un pseudo-type compilé en bool_t
  - `string` est compilé en `char *`. Attention, il est interdit de passer le pointeur NULL...
  - il existe un type opaque permettant d'obtenir des séquences d'octets (consulter la documentation à ce sujet)

- l'outil `rpcgen` permet de compiler une description RPCL en fichiers sources C
  - par convention les fichiers sources RPCL utilisent l'extension `.x`
- `rpcgen -a -NC fichier.x` permet d'obtenir l'ensemble des fichiers source C nécessaires à la partie cliente et serveur
  - l'encapsulation XDR est automatiquement générée...

- `rpcgen -a -NC fichier.x` va générer les fichiers :
  - en commun :
    - `fichier.h`
    - `fichier_xdr.c`
  - pour la partie cliente :
    - `fichier_client.c`
    - `fichier_clnt.c`
  - pour la partie serveur
    - `fichier_server.c`
    - `fichier_svc.c`

- le fichier `.h` contient la traduction en C des définitions RPCL, types et prototypes

- le fichier `_xdr.c` contient l'ensemble des fonctions de marshalling (le terme courant pour désigner la sérialisation XDR)

- le fichier `_svc.c` contient le programme principal (`main`) permettant d'enregistrer le service auprès de l'annuaire RPC (portmapper)

- le fichier `_server.c` contient une implémentation minimale de fonctions RPC (c'est ce fichier qui doit être modifié côté serveur : implémentation des calculs envisagés)

- le fichier `_clnt.c` contient les talons permettant les appels distants (via `clnt_call`)

- le fichier `_client.c` contient un squelette de client (`main`). C'est ce fichier qui est à modifier pour obtenir les effets recherchés côté client.

Les secrets de XDR

- Rappel : à l'instar du NBO pour les différentes couches réseau, la représentation des données échangées lors de RPC se doit d'être normalisée

- XDR (eXternal Data Representation) est une convention normalisée de marshalling (sérialisation ou linéarisation) des différentes données

- pour le programmeur il s'agit essentiellement d'une API contenant diverses fonctions

- le fonctionnement est simple :
  - le flux normalisé est représenté par un pointeur sur un objet de type XDR (fichier d'inclusion <rpc/types.h>)
  - le flux possède un attribut indiquant le sens des opérations :
    - XDR_ENCODE pour encoder vers le flux
    - XDR_DECODE pour décoder depuis le flux

- Il existe trois genres de flux :
  - les flux **fichiers** qui sont créés par appel à :

```
xdrstdio_create(XDR *x,FILE *f,
                enum xdr_op o);
```

  - les flux **en mémoire** qui sont créés par appel à :

```
xdrmem_create(XDR *x,char *m,
              u_int taille,enum xdr_op o);
```

  - les flux **génériques** qui sont créés par appel à :

```
xdrrec_create(XDR *x,u_int tEcr,
              u_int tLec,void *d,
              int (*lec()),int (*ecr()));
```

    - dans ce cas le type doit être positionné directement dans le champ x_op du flux

- la destruction d’un flux est obtenue par appel à :

```
xdr_destroy(XDR *x);
```

- à chaque type de donnée, une fonction d'encodage/décodage est fournie

  - `xdr_array`, `xdr_bool`, `xdr_bytes`, `xdr_char`, `xdr_double`, `xdr_enum`, `xdr_float`, `xdr_hyper`, `xdr_int`, `xdr_long`, `xdr_longlong_t`, `xdr_opaque`, `xdr_pointer`, `xdr_reference`, `xdr_short`, `xdr_string`, `xdr_u_char`, `xdr_u_hyper`, `xdr_u_int`, `xdr_u_long`, `xdr_u_longlong_t`, `xdr_u_short`, `xdr_union`, `xdr_vector`, `xdr_void`

- pour encore un entier (`int`) :

```
bool_t xdr_int(XDR *x,int *i);
```

- ou un `double` :

```
bool_t xdr_double(XDR *x,double *d);
```

- l'encodage/décodage des structures chaînées (par pointeurs) nécessite quelques précautions

  - la fonction xdr_pointer sert à encoder/décoder un objet désigné par un pointeur (le cas du pointeur nul est pris en compte) :

```
bool_t xdr_pointer(XDR *x,
                   char **pp,
                   u_int taille,
                   xdrproc_t encoder);
```

  - la fonction de codage sera appelée sous la forme :

```
encoder(x,*pp)
```

```c
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  XDR x; FILE *f; int i=0x12345678; char *s="bonjour";
  if (argc<2) {
    fprintf(stderr,"usage: %s file\n",argv[0]); exit(1);
  }
  f = fopen(argv[1],"w");
  if (f==NULL) {
    perror(argv[0]); exit(1);
  }
  ftruncate(f,0);
  xdrstdio_create(&x,f,XDR_ENCODE);
  xdr_int(&x,&i);
  xdr_string(&x,&s,7);
  xdr_destroy(&x);
  fclose(f);
  exit(0);
}
```

```c
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#include <stdlib.h>
#include <strings.h>

int main(int argc, char *argv[]) {
  XDR x; FILE *f; int i=0; char *s=malloc(10);
  bzero(s,10);
  if (argc<2) {
    fprintf(stderr,"usage: %s file\n",argv[0]); exit(1);
  }
  f = fopen(argv[1],"r");
  if (f==NULL) {
    perror(argv[0]); exit(1);
  }
  xdrstdio_create(&x,f,XDR_DECODE);
  xdr_int(&x,&i);
  xdr_string(&x,&s,7);
  xdr_destroy(&x);
  fclose(f);
  printf("i=%x\n",i);
  printf("s=%s\n",s);
  exit(0);
}
```

RPC (les secrets)

- il existe deux couches de programmation RPC
  - une couche dite « haute »
  - une couche dite « basse » autorisant un contrôle plus fin des mécanismes

La couche haute (serveur)

- du point de vue du serveur
  - deux fonctions :
    - l'enregistrement d'une fonction de service
  - la prise en compte de requêtes client

- `#include <rpc/rpc.h>`
  `bool_t registerrpc(u_long prognum,`
  `u_long version,u_long procnum,`
  `char *(*fonction)(),`
  `xdrproc_t encode,xdrproc_t decode);`

  permet d'enregistrer une `fonction` de service de numéro `procnum` associée à la `version` du service RPC de numéro `prognum`. La `fonction` sera appelée de sorte qu'elle

  - recevra en paramètre un pointeur sur les données envoyées
  - renverra un pointeur sur la valeur de retour
  - lesdites données seront encodées et décodées via les fonctions associées

- Attention, cette fonction n'effectue l'enregistrement du service qu'en UDP…

- `#include <rpc/rpc.h>`
  `void svc_run();`

  cette fonction :

  - boucle infinie permettant d'attendre des requêtes client

  - effectue la distribution vers les fonctions de service préalablement enregistrées

- Attention, du **côté serveur** les fonctions de service sont des fonctions :
  - **recevant un seul paramètre, un pointeur** sur un objet correspondant aux données
    - nécessite donc en général l'encapsulation dans une structure
  - **retournant un pointeur** sur un type correspondant aux données attendues
    - en général ce pointeur est en zone statique (sinon problème de gestion mémoire)

La couche haute (client)

- du point de vue du client, une fonction :

```
int callrpc(char *machine,u_long prognum,
            u_long version,u_long procnum,
            xdrproc_t encode,void *in,
            xdrproc_t decode,void *out);
```

  permet d'appeler la fonction de service de numéro `procnum` dans la `version` indiquée du service RPC de numéro `prognum`

- les arguments de la fonction de service seront encodés via la fonction d'encodage qui recevra l'argument `in`

- les résultats de la fonction de service seront décodés via la fonction de décodage qui recevra l'argument `out`

- cette fonction appelle le service UDP

Un exemple

- dans cet exemple :
    - le client appelle une procédure distante en passant une chaîne de caractère
    - laquelle est transformée à distance en convertissant les caractères minuscules en majuscules puis le résultat est retourné
    - au client qui affiche simplement le résultat

```c
#include <rpc/rpc.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

bool_t myencoding(XDR *x,char **m) {
  return xdr_string(x,m,100);
}

char **echo(char **v) { // convert
  char *c;
  for (c=*v;*c;c++) *c = toupper(*c);
  return v;
}

int main(int argc,char *argv[]) {
  int r;
  pmap_unset(0x87654321,1);
  if (argc==2) exit(0); // "stops" (unregister the service, only)
  r = registerrpc(0x87654321,1,1,echo,myencoding,myencoding);
  if (r==-1) { perror("registering failed\n"); exit(1); }
  svc_run();
  exit(0);
}
```

```c
#include <rpc/rpc.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>

bool_t myencoding(XDR *x,char **m) {
  return xdr_string(x,m,100);
}

char *echo(char *machine,char *m) {
  int r; static char s[101], *ss = s;
  bzero(ss,101);
  r = callrpc(machine,0x87654321,1,1,myencoding,&m,myencoding,&ss);
  if (r != RPC_SUCCESS) { clnt_perrno(r); return 0; }
  return ss;
}

int main(int argc,char *argv[]) {
  char *s;
  if (argc<3) {
    fprintf(stderr,"usage: %s host string\n",argv[0]); exit(1);
  }
  s = echo(argv[1],argv[2]);
  printf("%s\n",s);
  exit(0);
}
```

La couche basse

- dans ce mode de programmation, on contrôle :
  - le mode d'enregistrement du service (TCP ou UDP)
  - la fonction de distribution des requêtes (dispatch)
  - divers paramètres concernant la liaison

- les principales fonctions serveur sont :
  - `svc_create()` pour créer et enregistrer un service TCP ou UDP
    - `svcudp_create()` pour créer un descripteur de service UDP
    - `svctcp_create()` pour créer un descripteur de service TCP
    - `svc_register()` pour enregistrer un service déjà créé
  - `svc_destroy()` pour supprimer un descripteur de service
  - `svc_getargs()` pour récupérer des arguments dans le dispatch
  - `svc_sendreply()` pour renvoyer un résultat depuis le dispatch

- les principales fonctions client sont :
  - `clnt_create()` pour créer un descripteur client TCP ou UDP
    - `clntudp_create()` pour créer un descripteur client udp
    - `clnttcp_create()` pour créer un descripteur client tcp
    - `clnt_destroy()` pour détruire un descripteur client
  - `clnt_control()` pour paramétrer le client (liaison et service)
  - `clnt_call()` pour appeler une procédure distante

- On notera l’existence d’un mécanisme générique d’authentification pour les RPC
  - sa mise en œuvre n’est très compliquée
  - il existe deux schémas pré-définis :
    - AUTH_UNIX (en gros uid/gid)
    - AUTH_DES (cryptage mot de passe Unix)