APPLICATIONS DISTRIBUÉES CLIENT/SERVEUR:

XDR / RPC

Laurent PERROTON

ESIEE

Mai 2016

Plan

- L'informatique distribuée : outils de développement spécifiques
- Filtres `XDR`
- RPC

Informatique distribuée: mais qu'est-ce que c'est !?

Hier...

- **Multitude** d'utilisateurs
- **Pénurie** de ressources

Problématique: arbitrer l'allocation/le partage des ressources

L'informatique d'aujourd'hui :

- **Abondance** de ressources indépendantes les unes des autres

⇒ Pas de centralisation de l'administration des ressources

⇒ Comment trouver la bonne ressources ?

⇒ Accessibilité des ressources ?

⇒ Partage ?

⇒ Comment coopérer efficacement ?

Informatique distribué:

gestion efficace de ressources coopérantes

- Problématique différente
- Algorithmique différente
- Outils de développement différents

Les "socket"

les `socket` BSD

- Le premier "outils" de développement réseau permettant de gérer des ressources distribuées
- Standard *de fait*
- Outils de très bas niveau *(échange de* **bloc d'octets***)*
- Lourd à programmer *(nombreux paramètres, fonctions, API...)*
- Un protocole de communication *(type de données, format des messages, actions possibles...)* particulier doit etre défini pour chaque ressources
- Modèle par **passage de message**

$\Rightarrow$ *Nécessité de modèles de plus haut niveau d'abstraction pour les applications distribués complexes*

Applications Client/Serveur et RPC

Concept Client / Serveur

- Des **actions** sont associés à une ressources
- Ces actions sont mises à disposition des **clients** par le **serveur** sous forme de **services**
- Modèle indépendant de la couche de communication

RPC/XDR : Outils pour faciliter le développement d'applications client/serveur

- Basé sur une extension de la notion de **procédure** *(Remote Procedure Call)*
- Analyse d'une application similaire à l'approche classique "procédurale"
- portabilité, souplesse, concision et simplicité du modèle
- N'est pas non plus une "solution universelle" au développement des applications distribuées

eXternal Data Representation

Exemple de communication de données entre systèmes différents

Emetteur :

```
#include <stdio.h>
main () {
  float x = 12.034;
  int n = -4321;
  fwrite(&x, sizeof(float), 1, stdout);
  fwrite(&n, sizeof(int), 1, stdout);
}
```

Recepteur :

```
#include <stdio.h>
main () {
  float x; int n;
  fread(&x, sizeof(float), 1, stdin);
  fread(&n, sizeof(int), 1, stdin);
  printf("recu flotant : %f entier : %d\n",x,n);
}
```

Entre systèmes différents :

```
intel% expediteur | ssh user@arm recepteur
recu flotant : 0.00000 entier : 37589750
intel%
```

Le même au format `xdr` :

Emetteur `xdr` :

```
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
main () {
  XDR sortie;
  float x = 12.034;
  int n = -4321;

  xdrstdio_create ( &sortie, stdout, XDR_ENCODE );
  xdr_int( &sortie, &n);
  xdr_float( &sortie, &x);
}
```

Recepteur `xdr` :

```
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
main () {
  XDR entree;
  float x; int n;

  xdrstdio_create ( &entree, stdin, XDR_DECODE );
  xdr_int( &entree, &n);
  xdr_float( &entree, &x);
  printf("recu flotant : %f entier : %d\n",x,n);
}
```

Tableau de `float`

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

main () {
   char buffer [1024]; /* buffer de translation xdr */
   XDR sortie, entree;
   float out[10]=
           {1.25, 78.13, 23.04, -145.32, 45.89, 32.03};
   float in[10], *fp;
   char *ptr; int nb_elts, nb_elts_max = 10, i;

   xdrmem_create(&sortie, buffer, 1024, XDR_ENCODE);
   xdrmem_create(&entree, buffer, 1024, XDR_DECODE);

   ptr = (char *) out; nb_elts = 6;
   xdr_array(&sortie, &ptr, &nb_elts, nb_elts_max,
             sizeof(float), xdr_float);
   ptr = (char *) in; nb_elts = 0;
   xdr_array(&entree, &ptr, &nb_elts, nb_elts_max,
             sizeof(float), xdr_float);
   for (i=0; i<nb_elts; printf("in[%d]=%f\n",i,in[i++]) );

   ptr = (char *) out; nb_elts = 6;
   xdr_array(&sortie, &ptr, &nb_elts, nb_elts_max,
             sizeof(float), xdr_float);
   fp = NULL; nb_elts = 0;
   xdr_array(&entree, &fp, &nb_elts, nb_elts_max,
             sizeof(float), xdr_float);
   for (i=0; i<nb_elts; printf("in[%d]=%f\n",i,fp[i++]) );
}
```

Structures

Définition d'une structure :

```
typedef struct {
   char nom [20];
   char prenom [30];
   float age;
} identite;
```

Filtre XDR correspondant :

```
bool_t xdr_identite(XDR *flot, identite *id) {
   char *nom, *prenom;
   nom = id -> nom; prenom = id -> prenom;

   return ( xdr_string(flot, &nom, 19)
         && xdr_string(flot, &prenom, 29)
         && xdr_float (flot, &id->age) );
}
```

- `xdr_string` prends comme paramètre un `char **`, il faut donc allouer des `char *` intermédiaires dans le filtre pour passer leurs **adresses** en paramètres aux filtres `xdr_string`.
- `xdr_float` prends comme paramètre un pointeur `float *`, donc pas besoin de faire cette manipulation
- La taille maximum passée en paramètre de `xdr_string` ne comprends pas l'octet de fin de chaine `'\0'`
- Filtre de chaine de prototype `xdrproc_t` :

  ```
  xdr_wrapstring (XDR *f, char **ch);
  ```

  Appel `xdr_string(f, ch, MAXUN.UNSIGNED);`

Liste chainée

Noeud de liste chainée :

```
typedef struct Noeud { int a; struct Noeud *s; } Liste;
```

Filtre `xdr` des noeuds :

```
bool_t xdr_Liste(XDR *flot, Liste *pl) {
  return (
    /* encode la donnee du noeud */
    xdr_int(flot, &pl->a ) &&
    /* Si dernier noeud (pl->s == NULL), encode FALSE
       Sinon, encode TRUE et appel
            xdr_reference(flot, &pl->s, taille, xdr_Liste) */
    xdr_pointer(flot, (void **) &pl->s, sizeof(Liste), xdr_Liste)
  );
}
```

Encodage au retour de la récursion : (Rifflet)

```
bool_t xdr_Liste(XDR *flot, Liste *pl) {

  /* Si dernier noeud (pl->s == NULL), encode FALSE
     Sinon, encode TRUE et appel
            xdr_reference(flot, &pl->s, taille, xdr_Liste) */

   if ( xdr_pointer(flot, (void **) &pl->s,
      sizeof(Liste), xdr_Liste) == FALSE ) return FALSE;

   /* encode la donnee du noeud */
   return xdr_int(flot, &pl->a );
}
```

RPC Haut niveau : C/S calcul de vecteur

Constantes, type et filtres `XDR` (`vect.h` commun aux C et S)

```
#define NB_PROG 0x22222222
#define NB_VERS 1
#define PROC_MOY 1
#define PROC_SOM 2

typedef struct { int taille, *vecteur; } vecteur;
typedef struct { vecteur v1, v2; } couple_vecteur;

bool_t xdr_vecteur ( XDR *flot, vecteur *v ) {
  return ( xdr_array(flot, &v->vecteur, &v->taille,
           1000, sizeof(int), xdr_int) );
}

bool_t xdr_couple_vecteur(XDR *flot,couple_vecteur *cv){
  return ( xdr_vecteur (flot, &cv->v1 ) &&
           xdr_vecteur (flot, &cv->v2 ) );
}
```

Procédures de calculs : (Serveur)

```
typedef struct { int taille, *vecteur; } vecteur;
typedef struct { vecteur v1, v2; } couple_vecteur;

/* somme de 2 vecteurs... */
vecteur *somme_vect ( couple_vecteur *pc ) {
  int i;
  for ( i=0; i<pc->v1.taille; i++ ){
     pc->v1.vecteur[i] += pc->v2.vecteur[i];
  }
  return ( &pc->v1 );
}

/* moyenne des termes d'un vecteur... */
static float moyenne;
float *moyenne_vect ( vecteur *v ) {
  int i;
  moyenne = 0;
  for ( i=0; i<v->taille; i++ ){
     moyenne += v->vecteur[i];
  }
  moyenne = moyenne / v->taille;
  return ( &moyenne );
}
```

Fonction `main()` du serveur :

```
main ( int argc, char ** argv ) {
  registerrpc(NB_PROG, NB_VERS, PROC_MOY, moyenne_vect,
              xdr_vecteur, xdr_float);
  registerrpc ( NB_PROG, NB_VERS, PROC_SOM, somme_vect,
              xdr_couple_vecteur, xdr_vecteur);
  svc_run ();
}
```

Code du client haut niveau

```
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#include "vect.h"

main ( int argc, char ** argv ) {
  int t1[] = { 2,5,7,12,-3,4,7,19};
  int t2[] = { -2,15,-3,0,-13,-5,8,7 };
  int t3[10], i;
  float f;
  vecteur v1, v2, v3, *pv;
  couple_vecteur c1, c2;

  v1.vecteur = t1; v1.taille = sizeof(t1)/sizeof(int);
  v2.vecteur = t2; v2.taille = sizeof(t2)/sizeof(int);
  v3.vecteur = t3; v3.taille = 0;
  c1.v1 = v1; c1.v2 = v2;

  callrpc("info20", NB_PROG, NB_VERS, PROC_MOY,
         xdr_vecteur, &v1, xdr_float, &f);
  printf ( "Moyenne v1 : %f\n", f );

  clnt_perrno(callrpc("info20", NB_PROG, NB_VERS, PROC_SOM,
         xdr_couple_vecteur, &c1, xdr_vecteur, &v3 ) );
  printf ( "Somme v1+v2 : " );
  for (i=0; i<v3.taille; printf("%d ",v3.vecteur[i++]));
  printf ( "\n");
}
```

RPC bas niveau: principe de la fonction `dispatch()`

Structure `struct svc_req`

```
struct svc_req {
    unsigned long rq_prog;
    unsigned long rq_vers;
    unsigned long rq_proc;
    struct opaque_auth rq_cred;
    caddr_t       rq_clntcred;
    SVCXPRT       *rq_xprt;
};
```

Code de `dispatch()`

```
void dispatch (struct svc_req *p_req, SVCXPRT *p_svc ) {
  switch (p_req -> rq_proc) {
  case NULLPROC : /* fct de # 0 : ping */
                  ......

                  return;
  case NUMPROC_1 : /* fct de # NUMPROC_1 */
                  ......

                  return;
  case NUMPROC_2 : /* fct de # NUMPROC_2 */
                  ......

                  return;
  default : svcerr_noproc ( p_svc );
            return;
  }
}
```

RPC bas niveau : le serveur

Fichiers inclus :

```
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#include <netinet/in.h>
#include <rpc/auth.h>
#include <rpc/svc.h>
extern vecteur *somme_vect   (couple_vecteur *p);
extern float   *moyenne_vect (vecteur *p);
```

Fonction `main()`

```
main ( int argc, char ** argv ) {
  SVCXPRT *p_svc;
  p_svc = svcudp_create (RPC_ANYSOCK);
  svc_register(p_svc,NB_PROG,NB_VERS,dispatch,IPPROTO_UDP);
  svc_run ();
}
```

Fonction `dispatch()`

```
void dispatch (struct svc_req *p_req, SVCXPRT *p_svc ) {
  vecteur v1, v2; int t1 [1000]; int t2 [1000];
  couple_vecteur c1; float f;
  v1.vecteur = t1; v2.vecteur = t2; c1.v1 = v1; c1.v2 = v2;

  switch (p_req -> rq_proc) {
  case 0 : /* fct de # 0 : ping */
    if (svc_sendreply(p_svc, xdr_void, NULL) == FALSE)
      svcerr_decode(p_svc);
    return;
  case PROC_MOY : /* fct moyenne d'un vecteur */
    if (svc_getargs(p_svc, xdr_vecteur, &v1) == FALSE) {
      svcerr_decode(p_svc);
      return;
    }
    if (svc_sendreply(p_svc, xdr_float, moyenne_vect(&v1)) == FALSE)
      svcerr_decode(p_svc);
    return;
  case PROC_SOM : /* fct somme de 2 vecteurs */
    if (svc_getargs(p_svc, xdr_couple_vecteur, &c1) ==FALSE){
      svcerr_decode(p_svc);
      return;
    }
    if (svc_sendreply(p_svc, xdr_vecteur, somme_vect(&c1)) == FALSE)
      svcerr_decode(p_svc);
    return;
  default : svcerr_noproc (p_svc);
      return;
  }
}
```

RPC bas niveau : le client

Fichiers inclus, variables :

```
#include <stdio.h>
#include <stdlib.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#include <netinet/in.h>
#include <rpc/auth.h>
#include <rpc/svc.h>
#include <rpc/clnt.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/time.h>

int t1[] = { 2,5,7,12,-3,4,7,19};
int t2[] = { -2,15,-3,0,-13,-5,8,7 };
int t3[10]; vecteur v1, v2, v3, *pv;
couple_vecteur c1, c2;
float f;

CLIENT *clnt;
struct sockaddr_in ad_srv;
int socket;
struct hostent *h;
struct timeval periode, temps_total;
```

Code `main()`

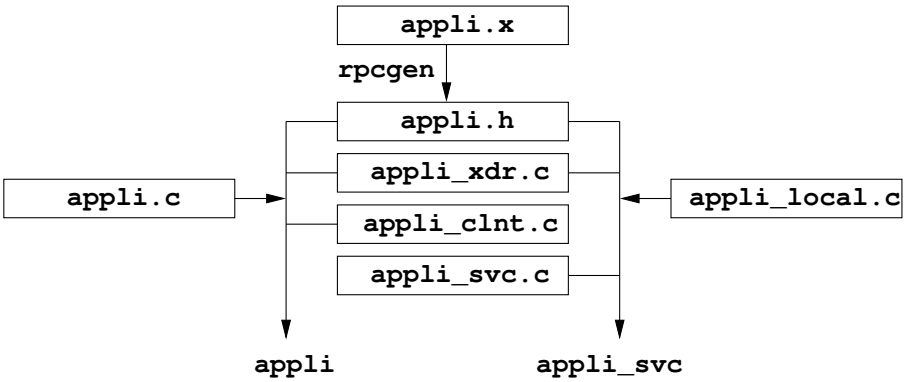
```
main ( int argc, char ** argv ) {
 v1.vecteur = t1; v1.taille = sizeof(t1)/sizeof(int);
 v2.vecteur = t2; v2.taille = sizeof(t2)/sizeof(int);
 v3.vecteur = t3; v3.taille = 0;
 c1.v1 = v1; c1.v2 = v2;

 h = gethostbyname ( argv[1] );
 ad_srv.sin_family = AF_INET;
 ad_srv.sin_port = 0;
 memcpy (&ad_srv.sin_addr.s_addr, h->h_addr, h->h_length);
 socket = RPC_ANYSOCK;

 periode.tv_sec = 6; periode.tv_usec = 0;
 clnt = clntudp_create ( &ad_srv, NB_PROG, NB_VERS,
                         periode, &socket );
 temps_total.tv_sec = 40; temps_total.tv_usec = 0;
 clnt_call ( clnt, PROC_MOY,
           xdr_vecteur, &v1, xdr_float, &f, temps_total);
 printf ( "Moyenne v1 : %f\n", f );

 if (clnt_call(clnt, PROC_SOM, xdr_couple_vecteur, &c1,
           xdr_vecteur, &v3, temps_total) != RPC_SUCCESS)
   clnt_perror(clnt , "Erreur appel somme");
 printf ( "Somme v1+v2 : " );
 for (i = 0;i<v3.taille;printf("%d ", v3.vecteur[i++]));
 printf ( "\n");
}
```

Compilateur de protocole `rpcgen`



Spécification RPCL : `appli.x`

```
const TAILLE = 1000;
typedef int vecteur<TAILLE>;
```

$\longrightarrow$

```
typedef struct {
        u_int vecteur_len;
        int *vecteur_val;
} vecteur;
```

```
struct couple_vecteur {
  vecteur v1;
  vecteur v2;
};

program NB_PROG {
      version NB_VERS {
              vecteur PROC_SOM ( couple_vecteur ) = 2;
              float PROC_MOY ( vecteur ) = 1;
      } = 1;
} = 0x22222222;
```

Modifications à apporter coté client (`appli.c`) :

- Respecter la convention de nommage de prototype de fonctions
- Création structure `CLIENT` (par exemple avec `clnt_create()`)
- Rajouter le 2nd argument aux fonctions RPC :

```
vecteur *proc_som_1(couple_vecteur *pc, CLIENT *) ..
```

Modifications à apporter coté serveur (`appli_local.c`) :

- Respecter la convention de nommage de prototype de fonctions
- Rajouter le 2nd argument aux fonctions RPC :

```
vecteur *proc_som_1_svc(couple_vecteur *pc, struct svc_req *) ..
```

Fichier inclus généré (ANSI C): `appli.h`

```
#define TAILLE 1000

typedef struct {
        u_int vecteur_len;
        int *vecteur_val;
} vecteur;
bool_t xdr_vecteur();

struct couple_vecteur {
        vecteur v1;
        vecteur v2;
};
typedef struct couple_vecteur couple_vecteur;
bool_t xdr_couple_vecteur();

#define NB_PROG ((u_long)0x22222222)
#define NB_VERS ((u_long)1)

#define PROC_SOM ((u_long)2)

extern vecteur *proc_som_1 (couple_vecteur *, CLIENT *);
extern vecteur *proc_som_1_svc(couple_vecteur *, struct svc_req *);

#define PROC_MOY ((u_long)1)

extern float *proc_moy_1 (vecteur *, CLIENT *);
extern float *proc_moy_1_svc (vecteur *, struct svc_req *);
```

Application client pour rpcgen : `appli.c`

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "appli.h"

main( int argc, char **argv ) {
  int t1[] = { 2,5,7,12,-3,4,7,19};
  int t2[] = { -2,15,-3,0,-13,-5,8,7 };
  vecteur v1, v2, *pv;
  couple_vecteur c1, c2;
  int i; float *pf;
  CLIENT *clnt;
  v1.vecteur_val=t1;v1.vecteur_len=sizeof(t1)/sizeof(int);
  v2.vecteur_val=t2;v2.vecteur_len=sizeof(t2)/sizeof(int);
  c1.v1 = v1; c1.v2 = v2;

  clnt = clnt_create ( argv[1], NB_PROG, NB_VERS, "udp" );

  pf = proc_moy_1 ( &v1, clnt );
  printf ( "Moyenne v1 : %f\n", *pf );

  pv = proc_som_1 ( &c1, clnt );
  printf ( "Somme v1+v2 : " );
  for (i=0;i<pv->vecteur_len;printf("%d ",pv->vecteur_val[i++]));
  printf ( "\n");
}
```

Fonction `svc_run()` : serveur RPC non bloquant

```
extern fd_set svc_fdset ;

svcrun () {
  fd_set ens_lecture;
  int nb_desc; /* Max open file per process */
  struct timeval delay;

  nb_desc = getdtablesize ();
  nb_desc = NOFILE; /* in <sys/param.h> */
  delay.tv_sec = 5;
  delay.tv_usec = 5;
  while ( 1 ) {
    ens_lecture = svc_fdset;
    switch ( select(nb_desc, (int *)&ens_lecture,
                                  NULL, NULL, &delay)) {
    case -1 :
      if ( errno == EBADF ) return; /* descriptor error */
      continue; /* else continue */
    case 0:
      printf ( "Temps d'attente ecoule... \n");
      /* custom processing */
      continue;
    default : printf ( "Requete recue... \n");
      svc_getreqset ( &ens_lecture );
    }
  }
}
```