Introduction aux Systèmes Distribués

XDR : eXternal Data Representation

Eric Cariou

Université de Pau et des Pays de l'Adour
Département Informatique

Eric.Cariou@univ-pau.fr

Introduction : XDR

- Lors d'une communication entre 2 machines distantes
  - Problème de l'hétérogénéité des systèmes et de la représentation des données
  - Exemples
    - Taille des types de base (long, short, int, double, float ...)
    - Ordre des octets dans codage binaire (*Big Endian* vs *Little Endian*)
  - On a déjà vu les fonctions de conversions « réseau » / « local » pour entiers longs (@IP) et courts (numéro port)
    - `htons, htonl, ntohs` et `ntohl`
- Nécessité de coder toute donnée, quelle que soit sa complexité et sa nature, indépendamment des systèmes
  - Protocole XDR : *eXternal Data Representation*
- XDR est utilisé lors des appels de procédures en RPC
- XDR peut aussi être utilisé seul (codage des données avant émission via une socket par exemple)

Introduction : RPC

- RPC : *Remote Procedure Call*
  - Appel de procédure à distance
  - Mécanisme permettant du coté serveur de proposer des services qui seront appelés par un client distant
    - Service = une fonction C
    - Coté client : appel d'une fonction locale qui aboutira à l'exécution de la fonction requise sur le serveur
      - Plus de nécessité de construire la mécanique de communication sous-jacente via les sockets
  - Toutes les données échangées pour réaliser l'appel à distance des fonctions sont codées via XDR pour gérer l'hétérogénéité entre les cotés client et serveur
- Origine de RPC et de XDR
  - Sun Microsystems dans les années 1980

Utilisation XDR & RPC

- Deux modes d'utilisation de XDR et RPC
  - Faire tout « à la main »
    - Définir les fonctions XDR pour coder les structures de données spécifiques à un programme
    - Ecrire les fonctions RPC cotés client et serveur permettant de réaliser la communication à distance et l'appel des fonctions
  - Utiliser le programme `rpcgen` pour générer automatiquement les fichiers définissant ces fonctions
    - `rpcgen` peut être utilisé seulement pour le codage XDR des structures de données d'un programme
      (sans avoir de « partie » RPC)

Principes généraux

- Principe général pour communication entre 2 éléments
  - Avant émission, on code les données
  - On envoie les données codées
  - En réception, on les décode
- Flots de codage et de décodage
  - Les données codées sont placées les unes à la suite des autres dans des flots XDR (XDR *streams*)
  - Les données contenues dans un flot XDR sont codées
    - Selon un standard de représentation
    - Standard interprétable par toute mise en oeuvre de XDR, quel que soit le système (matériel & logiciel)

Principe généraux

- Utilisation des flots
  - Support physique de flot (élément qui contiendra les données codées et à décoder)
    - Mémoire
    - Fichier
  - Un processus code les données dans le flot
    - Action de sérialisation
  - Un processus décode les données dans le flot
    - Action de désérialisation
  - A chaque type de données est associé une fonction de codage/décodage
    - Une seule fonction par type, on exécute les mêmes « actions », c'est le type de flot qui différe (il est en écriture ou en lecture)

Principes généraux

- (Dé)codage des types de données de toute nature
  - Types primitifs et chaîne
    - Il existe une liste de fonctions standards pour ces types
  - Structure définie par le programmeur
    - On définit une fonction dédiée pour chaque structure
    - Codage en série de chacun des membres en appelant la fonction associée à chaque type des membres
  - Tableaux
    - Fonction standard pour traiter tous les éléments du tableau
  - Pointeurs & autres ...
- Série de plusieurs données
  - On les code en série dans un certain ordre
  - Et on doit les décoder dans le même ordre

Types de flots XDR

- Trois sortes de flots
  - Flot créé en mémoire
  - Flot s'appuyant sur un fichier
  - Flot d'enregistrement
    - Enregistrement = bloc d'un ensemble de données
    - La lecture/écriture dans un flot d'enregistrement se fait par bloc de données plutôt que donnée par donnée
    - Un flot d'enregistrement peut être créé en mémoire ou dans un fichier
- Représentation d'un flot en C
  - Type `XDR`
  - Définitions des types et fonctions associés aux flots : `<rpc/rpc.h>` et fichiers associés

Types associés aux flots

- Enumération `xdr_op` qui contient 3 valeurs pour préciser le type d'un flot
  - XDR_ENCODE
    - Pour un flot en encodage (flot qui contiendra des données)
  - XDR_DECODE
    - Pour un flot en décodage (flot où on lira les données)
  - XDR_FREE
    - Pour un décodage libérant certaines zones mémoires utilisées
- Fonctions de codage/gestion des données/flots
  - Renverront presque toujours une valeur de type `bool_t`, énumération contenant 2 valeurs
    - FALSE
    - TRUE

Création de flot : mémoire

- Flot en mémoire
  - Doit réserver la zone mémoire associée au flot avant la création du flot
    - En codage, la zone mémoire contiendra les données codées
    - En décodage, on y lira les données codées
- Création du flot mémoire
  - `void xdrmem_create(`

```
    XDR *ptr_xdr,              pointeur sur le flot,
    void *adresse,             zone mémoire associée,
    unsigned int taille,       taille zone mémoire,
    enum xdr_op type);         type du flot
```

  - `*ptr_xdr` : référence sur un objet de type `XDR` existant qui est initialisé par l'appel de cette fonction

Création de flot : fichier

- Flot stocké dans un fichier
  - Doit créer ou ouvrir le fichier utilisé avant la création du flot
    - En codage, le fichier contiendra les données codées
    - En décodage, on y lira les données codées
  - Peut utiliser aussi `stdin` ou `stdout` pour utiliser les entrées et sorties standards
- Création du flot fichier
  -
```
void xdrstdio_create(
        XDR *ptr_xdr,          pointeur sur le flot,
        FILE *ptr_file         référence du fichier,
        enum xdr_op type);     type du flot
```

Création de flot : enregistrement

- Flot d'enregistrement
  - Un enregistrement = ensemble de données
  - A la création, on associe à ce type de flot
    - Un pointeur vers la ressource physique (fichier, zone mémoire ...) contenant les données
    - 2 fonctions pour lire et écrire dans cette ressource physique
    - Pour préciser si on définit un flot en codage ou décodage, on initialise directement le champ `x_op` de la structure `XDR`
- Création d'un flot d'enregistrement
  - `void xdrrec_create(`

```
        XDR *ptr_xdr,                pointeur sur le flot
        int taille_envoi,            taille tampon écriture
        int taille_reception,        taille tampon lecture
        void io_handle *,            pointeur sur la ressource
        int (* fct_lecture )(void *, char *, int),
        int (* fct_ecriture)(void *, char *, int));
```

Création de flot : enregistrement

- Détails fonctionnement flot enregistrement
  - Utilise 2 tampons en mémoire
    - Tampon d'écriture pour codage des données
    - Tampon de lecture pour décodage des données
  - Quand tampon d'écriture est plein
    - La fonction « `*fct_ecriture` » est appelée pour enregistrer le contenu du tampon dans la ressource physique
  - Quand tampon de lecture est vide
    - La fonction « `*fct_lecture` » est appelée pour remplir le contenu du tampon à partir de données lues dans la ressource physique
  - Taille des tampons
    - 4000 octets au minimum et par défaut
    - Si 0 en paramètre de `xddr_create` pour taille des tampons, prend cette taille par défaut

Création de flot : enregistrement

- Paramètres des 2 fonctions de lecture/écriture
  - `int fonction(`
    `void *io_handle,`
    `void *buffer,`
    `int taille);`
  - `io_handle` : pointeur sur l'identifiant de la ressource : la valeur passée lors de l'appel de `xdrrec_create`
  - `buffer` : le tampon mémoire où lire/écrire les données
  - `taille` : la taille de ce tampon
    - Nombre d'octets à lire pour écriture dans la ressource
    - Nombre d'octets maximum à écrire à partir de la ressource
  - Retourne le nombre d'octets lus ou écrits dans la ressource

Gestion des flots mémoire & fichier

- Récupérer la position courante
  - `u_int xdr_getpos(XDR *ptr_xdr)`
  - Retourne la position courante du flot
  - Si on a inséré plusieurs données sans changer à la main la position courante
    - Correspond à la taille occupée par les données dans le flot
    - Avec une communication réseau UDP/TCP : correspond à la taille des données (du flot) à transférer via une socket
- Modifier la position courante
  - `bool_t xdr_setpos(XDR *ptr_xdr, u_int pos)`
    - Se place dans le flot à la position `pos`
    - Renvoie TRUE si placement a pu être fait, FALSE si problème

Gestion des flots enregistrement

- Placement
  - `xdr_getpos` et `xdr_setpos` marchent en théorie pour tous les flots mais pas toujours en pratique avec les flots d'enregistrement
    - Si `xdr_getpos` ne permet pas de déterminer une position pertinente : renvoie -1
- En mode codage
  - `bool_t xdrrec_endofrecord(XDR *ptr_xdr, bool_t envoi_immediat)`
    - Marque la fin d'un enregistrement dans le flot `ptr_xdr`
    - Si `envoi_immediat` est à TRUE, force l'écriture immédiate dans la ressource physique du contenu du flot
    - Retourne TRUE si marquage a pu être fait, FALSE sinon

Gestion des flots enregistrement

- En mode décodage
  - `bool_t xdrrec_skiprecord(XDR *ptr_xdr)`
    - Se place au début du prochain enregistrement
      - En « zappant » les données non lues dans l'enregistrement courant
    - A utiliser aussi pour se placer au début du premier enregistrement
    - Retourne TRUE si le placement a pu être fait, FALSE sinon
  - `bool_t xdrrec_eof(XDR *ptr_xdr)`
    - Retourne FALSE s'il existe d'autres enregistrements après l'enregistrement courant
    - Retourne TRUE si l'on est arrivé au dernier enregistrement du flot

Suppression de flots

- Fonction de destruction d'un flot
  - `void xdr_destroy(XDR *ptr_xdr);`
  - Valable pour les 3 types de flots
  - Libère les ressource alloués à la création du flot
    - Le flot n'est alors plus utilisable

Types primitifs

- Gestion des types de base du langage C
  - Ensemble de primitives avec la forme générale pour un type *type*
    - `bool_t xdr_type(XDR *ptr_xdr, type *data)`
    - Retourne TRUE si le codage/décodage de la données `data` de type *type* a pu être réalisé dans le flux `ptr_xdr`, FALSE sinon
  - 10 fonctions de ce type
    - char : `bool_t xdr_char(XDR *ptr_xdr, char *data)`
    - short : `bool_t xdr_short(XDR *ptr_xdr, short *data)`
    - int : `bool_t xdr_int(XDR *ptr_xdr, int *data)`
    - long : `bool_t xdr_long(XDR *ptr_xdr, long *data)`
    - u_char : `bool_t xdr_u_char(XDR *ptr_xdr, u_char *data)`
    - u_short : `bool_t xdr_u_short(XDR *ptr_xdr, u_short *data)`
    - u_int : `bool_t xdr_u_int(XDR *ptr_xdr, u_int *data)`
    - u_long : `bool_t xdr_u_long(XDR *ptr_xdr, u_long *data)`
    - float : `bool_t xdr_float(XDR *ptr_xdr, float *data)`
    - double : `bool_t xdr_double(XDR *ptr_xdr, double *data)`
  - Selon systèmes également : `int32`, `int64`, `u_int32`, `u_int64` ...

Types programmeur

- Pour un type ou une structure défini par le programmeur
  - Doit aussi écrire une fonction de codage/décodage
  - On respectera les principes d'écriture pour un type `type`
    - Renvoie un `bool_t` pour préciser si le codage/décodage a pu être correctement réalisé (valeur TRUE) ou pas (valeur FALSE)
    - Nom de forme `xdr_`*`type`*
    - Deux paramètres
      - Premier : pointeur sur un objet XDR représentant le flot
      - Second : pointeur sur une valeur de type `type`
  - Note
    - Si on utilise à chaque fois un pointeur sur la donnée, c'est parce que la même fonction sert au codage et au décodage
      - Et on a besoin forcément d'un pointeur sur la variable pour décodage
  - Codage/décodage d'une structure
    - On code/décode en série l'ensemble de ses membres en s'appuyant sur les fonctions de codage/décodage des types des membres
    - Retourne FALSE si au moins le codage d'un membre a échoué

Exemples

- On veut coder 2 séries de données contenant chacune les variables suivantes
  - `int max`
  - `int min`
  - `double moy`
- 3 versions de cet exemple
  - Flot en mémoire avec encodage de 6 valeurs à la suite
  - Flot enregistrement en fichier avec encodage de 2 enregistrements de 3 valeurs
  - Flot en fichier avec encodage de 2 fois le contenu d'une structure

Exemple 1 : flot mémoire

- Coté codage
  - Fonction `codage_donnees`
    - Paramètre
      - `mem` : zone mémoire à utiliser pour stocker le flot
      - `taille` : taille de la zone
    - Retourne la taille occupée par les données ou -1 en cas d'erreur

```
int codage_donnees(char *mem, int taille) {
 XDR flot_xdr;
 int pos;
 // données à envoyer
 int min1, min2, max1, max2;
 double moy1, moy2;
 min1 = 5;  max1 = 12;  moy1 = 8.2;
 min2 = 9;  max2 = 13;  moy2 = 9.3;
```

Exemple 1 : flot mémoire

- Coté codage (suite)

```
// création du flot en encodage
xdrmem_create(&flot_xdr, mem, taille, XDR_ENCODE);

// codage des données
if (!xdr_int(&flot_xdr, &min1)) return -1;
if (!xdr_int(&flot_xdr, &max1)) return -1;
if (!xdr_double(&flot_xdr, &moy1)) return -1;
if (!xdr_int(&flot_xdr, &min2)) return -1;
if (!xdr_int(&flot_xdr, &max2)) return -1;
if (!xdr_double(&flot_xdr, &moy2)) return -1;

pos = xdr_getpos(&flot_xdr);
xdr_destroy(&flot_xdr);
return pos;
} // codage_donnees
```

Exemple 1 : flot mémoire

- Coté décodage
  - Fonction `decodage_donnees`
    - Paramètre
      - `mem` : zone mémoire contenant les données du flot
      - `taille` : taille de la zone
    - Retourne TRUE si le décodage s'est bien déroulé, FALSE sinon
    - Affiche les données décodées
  - `bool_t decodage_donnees(char *mem, int taille) {`

```
// données à décoder du flot
int min1, min2, max1, max2;
double moy1, moy2;

// création du flot en encodage
XDR flot_xdr;
xdrmem_create(&flot_xdr, mem, taille, XDR_DECODE);
```

Exemple 1 : flot mémoire

- Coté décodage (suite)

```
// décodage des données
if (!xdr_int(&flot_xdr, &min1)) return FALSE;
if (!xdr_int(&flot_xdr, &max1)) return FALSE;
if (!xdr_double(&flot_xdr, &moy1)) return FALSE;
if (!xdr_int(&flot_xdr, &min2)) return FALSE;
if (!xdr_int(&flot_xdr, &max2)) return FALSE;
if (!xdr_double(&flot_xdr, &moy2)) return FALSE;
xdr_destroy(&flot_xdr);

// affichage des données
printf("serie 1 : min = %d, max = %d, moy = %f\n"
        min1, max1, moy1);
printf("serie 2 : min = %d, max = %d, moy = %f\n",
        min2, max2, moy2);
return TRUE;

} // decodage_donnees
```

Exemple 1 : flot mémoire

- Code de test des fonctions

```
...
int nb;
bool_t res;
char buffer[TAILLEBUF]; // contiendra le flot
nb = codage_donnees(buffer, TAILLEBUF);
if (nb == -1) exit(1);
printf("*** codage réussi : nb = %d\n", nb);
res = decodage_donnees(buffer, TAILLEBUF);
if (!res) exit(1);
printf("*** fin du décodage\n");
...
```

- Trace d'exécution de ce code

```
*** codage réussi : nb = 32
serie 1 : min = 5, max = 12, moy = 8.200000
serie 2 : min = 9, max = 13, moy = 9.300000
*** fin du décodage
```

Exemple 1 : flot mémoire

- Commentaires
  - Les parties de code réalisant le codage et décodage des données sont strictement identiques
    - Seul l'initialisation du flot permet de préciser si on fait du codage ou décodage
  - La taille codée est de 32 octets
    - Dans le cas d'une transmission via des sockets, on n'aurait besoin de ne faire transiter que les 32 premiers octets du buffer mémoire

Exemple 2 : enregistrement

- Flot d'enregistrement dans un fichier
  - Fonctions de lecture/écriture associées au flot
    - Lecture/écriture dans fichier
  -
```
int lecture_fichier(FILE *fic, char *buffer,
                    int taille) {
  int nb_lus;
  nb_lus = fread(buffer, 1, taille, fic);
  if (nb_lus==0) return -1;
  return nb_lus;
}

int ecriture_fichier(FILE *fic, char *buffer,
                     int taille) {
  int nb_ecrits;
  nb_ecrits = fwrite(buffer, 1, taille, fic);
  if (nb_ecrits==0) return -1;
  return nb_ecrits;
}
```

Exemple 2 : enregistrement

- Coté codage
  - Même fonction `codage_donnees` que pour exemple précédent mais avec comme paramètre le nom du fichier
  - ```
    int codage_donnees(char *nomFic) {
     XDR flot_xdr;
     int pos;
     FILE *fic;
     // données à envoyer
     int min1, min2, max1, max2;
     double moy1, moy2;
     min1 = 5;  max1 = 12;  moy1 = 8.2;
     min2 = 9;  max2 = 13;  moy2 = 9.3;
     // ouverture du fichier
     fic = fopen(nomFic, "w");
     // ouverture du flot d'encodage
     flot_xdr.x_op = XDR_ENCODE;
     xdrrec_create(&flot_xdr, 0, 0, fic,
                   lecture_fichier, ecriture_fichier);
    ```
    ```

Exemple 2 : enregistrement

- Coté codage (suite)

```
// codage des données
// premier enregistrement : série 1
if (!xdr_int(&flot_xdr, &min1)) return -1;
if (!xdr_int(&flot_xdr, &max1)) return -1;
if (!xdr_double(&flot_xdr, &moy1)) return -1;
if (!xdrrec_endofrecord(&flot_xdr, TRUE))
    return -1;
// second enregistrement : série 2
if (!xdr_int(&flot_xdr, &min2)) return -1;
if (!xdr_int(&flot_xdr, &max2)) return -1;
if (!xdr_double(&flot_xdr, &moy2)) return -1;
if (!xdrrec_endofrecord(&flot_xdr, TRUE))
    return -1;

pos = xdr_getpos(&flot_xdr);
fclose(fic); xdr_destroy(&flot_xdr);
return(pos);
} // codage_donnees
```

Exemple 2 : enregistrement

- Coté décodage
  - Même fonction `decodage_donnees` que pour exemple précédent mais avec comme paramètre le nom du fichier
  -
```
bool_t decodage_donnees(char *nomFic) {
 FILE *fic;
 XDR flot_xdr;
 // données à décoder du flot, préalablement initialisées à 0
 int min1, min2, max1, max2;
 double moy1, moy2;
 min1 = 0;  max1 = 0;  moy1 = 0.0;
 min2 = 0;  max2 = 0;  moy2 = 0.0
 // ouverture du fichier
 fic = fopen(nomFic, "w");
 // ouverture du flot de décodage
 flot_xdr.x_op = XDR_DECODE;
 xdrrec_create(&flot_xdr, 0, 0, fic,
               lecture_fichier, ecriture_fichier);
```

Exemple 2 : enregistrement

- Coté décodage (suite)

```
// décodage des données
// on se place sur le premier enregistrement
if (!xdrrec_skiprecord(&flot_xdr))return FALSE;
if (!xdr_int(&flot_xdr, &min1)) return FALSE;
// on se place sur le second enregistrement (on zappe la fin du premier)
if (!xdrrec_skiprecord(&flot_xdr))return FALSE;
if (!xdr_int(&flot_xdr, &min2)) return FALSE;
if (!xdr_int(&flot_xdr, &max2)) return FALSE;
if (!xdr_double(&flot_xdr, &moy2)) return FALSE;
// affichage des données
printf("serie 1 : min = %d, max = %d, moy = %f\n"
          min1, max1, moy1);
printf("serie 2 : min = %d, max = %d, moy = %f\n",
          min2, max2, moy2);

fclose(fic);
xdr_destroy(&flot_xdr);
return TRUE;
} // decodage_donnees
```

Exemple 2 : enregistrement

- Code de test des fonctions

```
...
int nb;
bool_t res;
nb = codage_donnees("data.bin");
if (nb == -1) exit(1);
printf("*** codage réussi : nb = %d\n", nb);
res = decodage_donnees("data.bin");
if (!res) exit(1);
printf("*** fin du décodage\n");
...
```

- Trace d'exécution de ce code

```
*** codage réussi : nb = -1
serie 1 : min = 5, max = 0, moy = 0.000000
serie 2 : min = 9, max = 13, moy = 9.300000
*** fin du décodage
```

Exemple 2 : enregistrement

- Commentaires
  - Le `xdr_getpos` du codage du flot retourne -1
    - On ne connaît donc pas la taille totale des données codées
    - Pas forcément un problème car
      - Ce sont les fonctions de lecture/écriture qui font la lecture/écriture dans la ressource et à qui on précise la taille des données à écrire ou savent combien lire
      - En général, on traite un flot d'enregistrement avec une boucle de décodage qui s'arrête quand `xdrrec_eof` renvoie vrai
  - Taille du fichier data.bin : 40 octets
    - 8 octets de plus que pour flot mémoire : 4 octets supplémentaires sont ajoutés à chaque enregistrement pour le gérer

Exemple 2 : enregistrement

- Commentaires (suite)
  - Premier enregistrement décodé
    - On a bien passé le décodage de `max1` et `moy1` puisqu'ils sont restés à 0
  - Pour préciser si on est en codage/décodage
    - On positionne directement le champ `x_op` du flot XDR initialisé avec la valeur XDR_ENCODE ou XDR_DECODE
  - Fonctions `lecture/ecriture_fichier`
    - Le pointeur de fichier passé en premier paramètre est celui passé en paramètre `iohandle` du `xdrrec_create`

Exemple 3 : fichier et structure

- On définit une structure qui code les 3 valeurs
  - ```
    struct stats {
      int min;
      int max;
      double moy;
    };
    ```
- La fonction de codage/décodage associée
  - ```
    bool_t xdr_stats(XDR *ptr_xdr, struct stats *data)
    {
      return(xdr_int(ptr_xdr, &data -> min) &&
             xdr_int(ptr_xdr, &data -> max) &&
             xdr_double(ptr_xdr, &data -> moy));
    }
    ```
  - On code/décode en série les 3 membres en appelant les primitives de codage/décodage des `int` et `double`

Exemple 3 : fichier et structure

- Coté codage
  - Même fonction `codage_donnees` que pour exemple 2
  - `int codage_donnees(char *nomFic) {`

```
XDR flot_xdr;
FILE *fic;
int pos;
// données à envoyer
struct stats stats1, stats2;
stats1.min = 5; stats1.max = 12; stats1.moy = 8.2;
stats2.min = 9; stats2.max = 13; stats2.moy = 9.3;

// ouverture du fichier et du flot
fic = fopen(nomFic, "w");
xdrstdio_create(&flot_xdr, fic, XDR_ENCODE);
```

Exemple 3 : fichier et structure

- Coté codage (suite)
  - // codage des 2 structures

```
if (!xdr_stats(&flot_xdr, &stats1)) return -1;
if (!xdr_stats(&flot_xdr, &stats2)) return -1;

pos = xdr_getpos(&flot_xdr);
fclose(fic);
xdr_destroy(&flot_xdr);
return(pos);
} // codage_donnees
```

Exemple 3 : fichier et structure

- Coté décodage
  - Même fonction `decodage_donnees` que pour exemple 2
  - ```
    bool_t decodage_donnees(char *nomFic) {
     XDR flot_xdr;
     FILE *fic;
     // données à décoder du flot
     struct stats stats1, stats2;
     // ouverture du fichier et du flot
     fic = fopen(nomFic, "r");
     xdrstdio_create(&flot_xdr, fic, XDR_DECODE);
     // décodage
     if (!xdr_stats(&flot_xdr, &stats1)) return FALSE;
     if (!xdr_stats(&flot_xdr, &stats2)) return FALSE;
    // affichage des données
     printf("serie 1 : min = %d, max = %d, moy = %f\n"
              stats1.min, stats1.max, stats1.moy);
     printf("serie 2 : min = %d, max = %d, moy = %f\n",
              stats2.min, stats2.max, stats2.moy);
     fclose(fic); xdr_destroy(&flot_xdr);
     return TRUE; }
    ```
    ```

Exemple 3 : fichier et structure

- Trace d'exécution du code de test exemple 2

```
*** codage réussi : nb = 32
serie 1 : min = 5, max = 12, moy = 8.200000
serie 2 : min = 9, max = 13, moy = 9.300000
*** fin du décodage
```

- Commentaires
  - Taille du fichier data.bin fait 32 octets et 32 octets écrits
    - Comme pour exemple 1 : pas de surcharge due à la structure
  - Code de codage/décodage bien plus compact et plus « sûr »
    - On est sûr de lire tous les membres et dans le bon ordre

Types et structures avancés

- Fonctions pour gérer des types et des structures plus complexes
  - Chaînes de caractères
  - Données « opaques »
  - Enumération
  - Tableaux
    - Octets
    - Taille fixe
    - Taille variable
  - Pointeurs
    - Simple : sur une donnée
    - Complexe : liste chaînée, arbre ...
  - Union avec discriminant

Enumérations

- Codage d'un élément de type énumération
  - `bool_t xdr_enum(`
    `XDR *ptr_xdr,`
    `enum_t *data);`
    - `enum_t` : type générique de tous les types énumérations
- Codage d'un booléen de type `bool_t`
  - Cas particulier de codage d'énumération
  - `bool_t xdr_bool(`
    `XDR *ptr_xdr,`
    `bool_t *data);`

Chaîne de caractères

- En C, une chaîne est
  - Un tableau/une suite de caractères dont la fin est précisée par le caractère de code ASCII de valeur 0
  - Pas forcément ce codage pour tous systèmes/langages
- Fonction dédiée au codage des chaînes
  - `bool_t xdr_string(`
    `XDR *ptr_xdr,` pointeur sur le flot,
    `char **ptr,` pointeur sur la chaine,
    `u_int longueur_max);` longueur max de la chaîne
- En codage
  - Code la chaine référencée par `*ptr`
  - Taille max de la chaine
    - Si on essaye d'encoder une chaine de taille supérieure à cette valeur, le codage retourne FALSE

Chaîne de caractères

- En décodage
  - Deux cas selon la valeur de `*ptr`
    - Valeur NULL passée
      - L'espace mémoire contenant la chaîne décodée est alloué dynamiquement
      - Avec une taille d'au plus `longueur_max` caractères, sinon le décodage retourne FALSE
    - `*ptr` a une valeur différente de NULL
      - Contient l'adresse du pointeur de chaîne (`*ptr`) qui correspond à une zone mémoire déjà réservée pour décoder la chaîne
      - Cette zone a une taille d'au plus `longueur_max` caractères
  - Dans les 2 cas
    - `*ptr` référence le pointeur sur la chaîne
    - La chaîne décodée suit le standard de codage du C
      - Suite de caractères se terminant par le caractère de code ASCII 0

Tableaux d'octets

- Gestion des tableaux d'octets
  - `bool_t xdr_bytes(`

    `XDR *ptr_xdr,` pointeur sur le flot,

    `char **ptr,` pointeur sur le pointeur du tableau,

    `u_int *ptr_lg,` pointeur sur la taille du tableau,

    `u_int lg_max);` taille max du tableau
  - En codage
    - `*ptr_lg` contient la taille du tableau à coder
  - En décodage
    - `*ptr_lg` contient la taille du tableau décodé
    - Si `*ptr` est NULL
      - Allocation dynamique du tableau comme pour les chaînes
      - `*ptr` référence la zone mémoire allouée
    - Sinon, `*ptr` référence une zone mémoire déjà allouée
    - Décodage valide pour un tableau de taille d'au plus `lg_max`

Tableaux d'éléments

- Codage/décodage d'un tableau d'un certain type d'éléments
  - `bool_t xdr_array(`

`XDR *ptr_xdr,`	pointeur sur le flot,
`void **ptr,`	adresse du tableau,
`u_int *nb_elts,`	pointeur sur nombre éléments
`u_int nb_elt_max,`	nombre max d'éléments
`u_int taille_elt,`	taille d'un élément
`xdrproc_t xdr_fonc);`	pointeur sur la fonction XDR

  - En codage
    - Codage de `*nb_elts` éléments d'une taille `taille_elt` dans le tableau d'adresse `*ptr`
  - En décodage
    - Si `*ptr` est égal à NULL, allocation dynamique d'un tableau contenant au plus `nb_elt_max`
    - `*nb_elts` contient le nombre d'éléments décodés
  - Chaque élément est (dé)codé par `xdr_fonc`, fonction XDR de type : `typedef bool_t (*xdrproc_t) (XDR *, void *)`

Tableaux d'éléments

- Codage/décodage d'un tableau d'un certain type d'éléments dont le nombre d'éléments est connu
  - Variante de `xdr_array` mais sans avoir besoin de préciser un nombre max d'éléments
  - `bool_t xdr_vector(`

```
    XDR *ptr_xdr,          pointeur sur le flot,
    void *ptr,             adresse du tableau,
    u_int nb_elts,         nombre d'éléments du tableau
    u_int taille_elt,      taille d'un élément
    xdrproc_t xdr_fonc);   pointeur sur la fonction XDR
```

  - En décodage
    - Pas d'allocation dynamique lors du décodage
    - Pas de précision du nombre d'éléments décodés car il est connu par principe si on utilise cette fonction

Tableaux : exemple

- Modification de l'exemple précédent – version 3
  - Codage d'un tableau de 2 éléments `struct stats`
  - Coté codage
    -
```
int codage_donnees(char *nomFic) {
 XDR flot_xdr;
 FILE *fic;
 int pos;

 // données à envoyer
 struct stats tab_stats[2];
 tab_stats[0].min = 5;
 tab_stats[0].max = 12;
 tab_stats[0].moy = 8.2;

 tab_stats[1].min = 9;
 tab_stats[1].max = 13;
 tab_stats[1].moy = 9.3;
```

Tableaux : exemple

- Coté codage (suite)

```
// pointeur sur le tableau et nombre éléments
void *ptr_stats = (void *)tab_stats;
int lg = 2;
int lg_max = 2;

// ouverture du fichier et du flot
fic = fopen(nomFic, "w");
xdrstdio_create(&flot_xdr, fic, XDR_ENCODE);

// codage du tableau
if (!xdr_array(&flot_xdr, &ptr_stats, &lg, lg_max,
     sizeof(struct stats), xdr_stats)) return -1;

pos = xdr_getpos(&flot_xdr);
fclose(fic);
xdr_destroy(&flot_xdr);
return(pos);
} // codage_donnees
```

Tableaux : exemple

- Coté décodage
  - bool_t decodage_donnees(char *nomFic) {

```
XDR flot_xdr;
FILE *fic;
int i;
// gestion des données à décoder du flot
// on réserve une zone de 5 éléments
struct tab_stats[5];
void *ptr_stats = (void *)tab_stats;
int lg = 2;                    // valeur non significative puisque décodage
int lg_max = 5;

// ouverture du fichier et du flot
fic = fopen(nomFic, "r");
xdrstdio_create(&flot_xdr, fic, XDR_DECODE);

// décodage du tableau
if (!xdr_array(&flot_xdr, &ptr_stats, &lg, lg_max,
     sizeof(struct stats), xdr_stats)) return FALSE;
```

Tableaux : exemple

- Coté décodage (suite)
  - // affichage des données
    // lg contient le nombre d'éléments décodés dans le flot

```
for (i = 0, i < lg; i++)
   printf("serie %d : min = %d, max = %d, moy = %f\n",
          i+1, tab_stats[i].min, tab_stats[i].max,
          tab_stats[i].moy);

 fclose(fic); xdr_destroy(&flot_xdr);
 return TRUE;
}
```

- Trace d'exécution
  -

```
*** codage réussi : nb = 36
serie 1 : min = 5, max = 12, moy = 8.200000
serie 2 : min = 9, max = 13, moy = 9.300000
*** fin du décodage
```

  - Taille de 36 octets au lieu de 32
    - 2 octets sont rajoutés par élément du tableau

Pointeurs simples

- Codage et décodage d'une valeur pointée
  - En décodage, on récupère un pointeur sur la valeur qui a été décodée
  - `bool_t xdr_reference(`

```
    XDR *ptr_xdr,
    void **ptr,
    u_int taille,
    xdrproc_t xdr_fonc);
```

pointeur sur le flot,
pointeur sur le pointeur,
taille de la donnée pointée,
fonction XDR pour codage de la donnée pointée

    - En décodage, allocation dynamique de la donnée si `*ptr` est égal à NULL

Pointeurs « complexes »

- Fonction `xdr_pointer`
  - `bool_t xdr_pointer(`

```
    XDR *ptr_xdr,
    void **ptr,
    u_int taille,
    xdrproc_t xdr_fonc);
```

pointeur sur le flot,
pointeur sur le pointeur,
taille de la donnée pointée,
fonction XDR pour codage de la donnée pointée

  - Fonctionnement identique à `xdr_reference` pour codage d'un pointeur et de la donnée pointée
  - Sauf dans le cas où le pointeur est égal à NULL
    - `xdr_reference` retourne FALSE sans plus de traitement
    - `xdr_pointer` retourne FALSE mais code en plus la valeur NULL dans le flot
  - En pratique, `xdr_pointer` sert à coder des structures complexes utilisant des pointeurs comme des listes chaînées, des arbres ... (Voir TD pour exemple d'utilisation)

Unions

- Codage des unions avec discriminants
  - `bool_t xdr_union(`
    `XDR *ptr_xdr,`
    `enum_t *ptr_discrim,`
    `void *adresse,`
    `struct xdr_discrim *ptr_select,`
    `xdrproc_t xdr_defaut);`
  - Avec la structure `xdr_discrim` pour associer une fonction de codage XDR à un type de l'union
    - `struct xdr_discrim {`
      `int value;`
      `xdrproct_t proc;`
      `}`
    - Valeur pour type non reconnu
      - `#define NULL_xdrproc_t ((xdrproc_t)0)`
  - Voir TD pour détails

Donnés opaques et type void

- Codage données opaques
  - Données dont on ne connait pas la structure exacte
  - `bool_t xdr_opaque(`

    `XDR *xdr_ptr,` pointeur sur le flot,

    `void *ptr,` pointeur sur la zone des données

    `u_int taille);` taille de la zone de données
  - Exemple d'utilisation de cette fonction
    - Un serveur envoie un bloc de donnée d'identification à chaque client
    - Client envoie ce bloc avec sa requête pour être identifié
    - La structure et le contenu exacts du bloc d'identification n'ont besoin d'être connus que du serveur
- Codage type `void`
  - `bool_t xdr_void()`
    - Retourne toujours TRUE
    - Utiliser par les mécanismes RPC, pas d'utilité sinon