

TD 7 - Remote Procedure Call (RPC)

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com

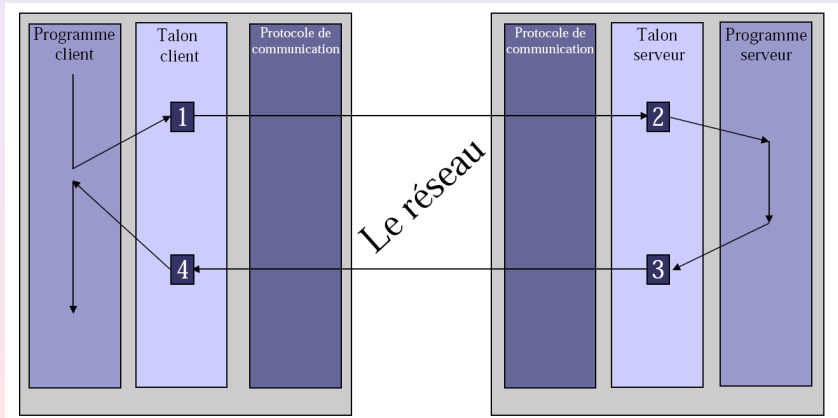
Plan de la présentation

- 1 Objectifs
- 2 Principe général de fonctionnement
- 3 Un cas particulier : Les RPC sous Linux
- 4 Un outils : rpcgen
- 5 Un exemple
- 6 Concepts avancés
- 7 Les RPC en Java

Objectifs

- Souvent, la communication par socket = invocation de commande à distance.
- Problèmes :
 - Lourd à programmer : Encodage des données (paramètres, résultats, ...), identification du serveur, du protocole, etc.
 - Pas naturel.
 - Élaboration d'un énorme *switch* au niveau du serveur.
- Retrouver la sémantique habituelle de l'appel de procédure :
 - sans se préoccuper de la localisation de la procédure,
 - sans se préoccuper du traitement des défaillances.
- Les difficultés :
 - Appel de procédures locales :
 - Appelant et appelé dans le même espace virtuel : même mode de pannes, appel et retour fiable.
 - Appel de procédures distantes :
 - Appelant et appelé dans deux espaces virtuels : mode de pannes indépendant, réseau non fiable, temps de réponse.

Principe général de fonctionnement



Les RPC sous Linux

- Protocole défini par SUN :
 - Il est à la base de l'implémentation de NFS.
 - Il est Open Source.
 - Il utilise le protocole XDR pour les échange de données (transport des arguments et du résultat).
- Fonctionnement :
 - Au niveau serveur :
 - un processus démon attend des connexions (*portmap*).
 - il détermine le programme p qui contient la procédure (qui s'est au préalable fait référencer).
 - le programme p décode les paramètres, exécute la procédure et encode le résultat.
 - le démon retourne le résultat.
 - Au niveau client, le talon du client va :
 - déterminer le numéro du programme (public : $0x20000000$ à $0x3fffffff$).
 - déterminer la version du programme à utiliser.

Développements

- Il existe trois façon de développer des programmes utilisant des RPC :
 - Utiliser les fonctions de la couche *intermédiaire*.
 - Utiliser les fonctions de la couche *base*.
 - Utiliser le compilateur *rpcgen*.

La couche intermédiaire

- Elle comporte peu de fonctions :

```
int registerrpc(unsigned long prg, unsigned long ver,  
               unsigned long proc, void *(*f)(),  
               xdrproc_t xdr_param, xdrproc_t xdr_result)
```

```
void pmap_unset (unsigned int prog, unsigned int ver)
```

```
void svc_run()
```

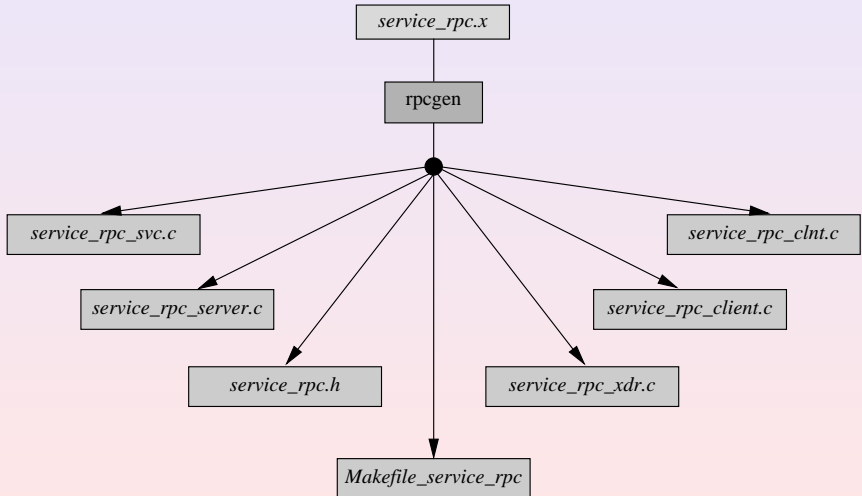
```
int callrpc(char *host, unsigned long prog, unsigned long vers,  
            unsigned long proc, xdrproc_t xdr_param, void* param,  
            xdrproc_t xdr_result, void* result)
```

- Mais :
 - le programmeur est limité dans la configuration du système (udp, pas d'authentification possible).
 - le programmeur doit développer l'encodage et le décodage.

La couche basse

- C'est un ensemble complet de fonctions
- Mais son utilisation est beaucoup plus complexe que la couche intermédiaire :
 - Elle possède plus de 20 fonctions
- À n'utiliser que :
 - lorsque le protocole de communication et les délais de temporisation de la couche intermédiaire ne sont pas satisfaisant.
 - lorsqu'on veut développer des RPC asynchrones.
 - lorsqu'on veut authentifier le client.

Le compilateur RPCGEN...



Le compilateur *rpcgen* et le langage de description RPCL

- Utilisation de *rpcgen* :

- `rpcgen service_rpc.x` → `.h _xdr.c _svc.c _clnt.c`
- `rpcgen -a service_rpc.x` → `_client.c _server.c Makefile`
- `rpcgen -c service_rpc.x` → `_xdr.c`
- `rpcgen -h service_rpc.x` → `.h`
- `rpcgen -l service_rpc.x` → `_clnt.c`
- `rpcgen -s transport service_rpc.x` → `_svc.` (tcp ou udp)
- `rpcgen -m transport service_rpc.x` → `_clnt.c` sans `main()` (tcp ou udp).

- Les fichiers `.x` ont la structure suivante :

```
[Definition des constantes]
[Definition des types]
program NOM_PROGRAMME {
    [version NOM_VERSION {
        [type_resultat nom_procedure
         (type_du_parametre) = numero_procedure:]
        ] = numero_version]
    } = numero_programme;
```

Le langage de description RPCL

- Les constantes :
const identificateur = valeur

- Les types :

```
struct nom_du_type {  
    type attribut;  
}
```

- ou

```
typedef type nom_du_type;
```

- Cas particulier des tableaux et chaînes de caractères :

```
typedef int vecteur <1000>
```

```
typedef string chaine <255>
```

L'exemple Helloworld en C sous Linux

- Le fichier de description *helloworld.x*

```
typedef string chaine<255>;  
  
program HELLO_WORLD_PROG {  
    version HELLO_WORLD_VERSION_1 {  
        void hello_world_null(void)=0;  
        chaine hello_world(chaine)=1;  
    }=1;  
} = 0x222222220;
```

- `rpcgen helloworld.x`
 - `helloworld.h`
 - `helloworld_xdr.c`
 - `helloworld_svc.c`
 - `helloworld_clnt.c`
- `rpcgen -a helloworld.x`
 - `Makefile`
 - `helloworld_server.c`

helloworld_server.c

```
/*  
 * This is sample code generated by rpcgen.  
 * These are only templates and you can use them  
 * as a guideline for developing your own functions.  
 */  
#include "helloworld.h"  
  
void * hello_world_null_1_svc(void *argp, struct svc_req *rqstp) {  
    static char * result;  
    printf ("Ping\n");fflush (stdout);  
    return (void *) &result;  
}  
  
chaîne * hello_world_1_svc(chaine *argp, struct svc_req *rqstp) {  
    static chaîne result;  
    static char tab[255];  
    /*  
     * insert server code here  
     */  
    result=tab;  
    strcpy (result, "Hello ");  
    strcat (result, *argp);  
    printf ("Result: %s\n", *argp);  
    printf ("Result: %s\n", result);  
    return &result;  
}
```

helloworld_client.c

```
/*  
 * This is sample code generated by rpcgen.  
 * These are only templates and you can use them  
 * as a guideline for developing your own functions.  
 */  
  
#include "helloworld.h"  
  
void hello_world_prog_1(char *host) {  
    CLIENT *clnt;  
    void *result_1;  
    char *hello_world_null_1_arg;  
    chaine *result_2;  
    chaine hello_world_1_arg;  
#ifndef DEBUG  
    clnt = clnt_create (host, HELLO_WORLD_PROG, HELLO_WORLD_VERSION_1, "udp");  
    if (clnt == NULL) {  
        clnt_pcreateerror (host);  
        exit (1);  
    }  
#endif /* DEBUG */  
    result_1 = hello_world_null_1((void*)&hello_world_null_1_arg, clnt);  
    if (result_1 == (void *) NULL) {  
        clnt_perror (clnt, "call failed");  
    }  
  
    result_2 = hello_world_1(&host, clnt);  
    if (result_2 == (chaine *) NULL) {  
        clnt_perror (clnt, "call failed");  
    }  
    printf ("%s\n", *result_2);  
#ifndef DEBUG  
    clnt_destroy (clnt);  
#endif /* DEBUG */  
}  
  
int main (int argc, char *argv[]) {  
    if (argc < 2) {  
        printf ("usage: %s server_host\n", argv[0]);  
    }  
}
```

Le programme HelloWorld

- Ne pas oublier de dé-enregistrer le processus serveur.

```
#include "helloworld.h"

main (){
    pmap_unset (HELLO_WORLD_PROG, HELLO_WORLD_VERSION_1);
}
```

- Vous pouvez télécharger l'ensemble des sources du programme à l'adresse

<http://litis.univ-lehavre.fr/~duvallet/enseignements/enseignements-MATIS-fr.php>

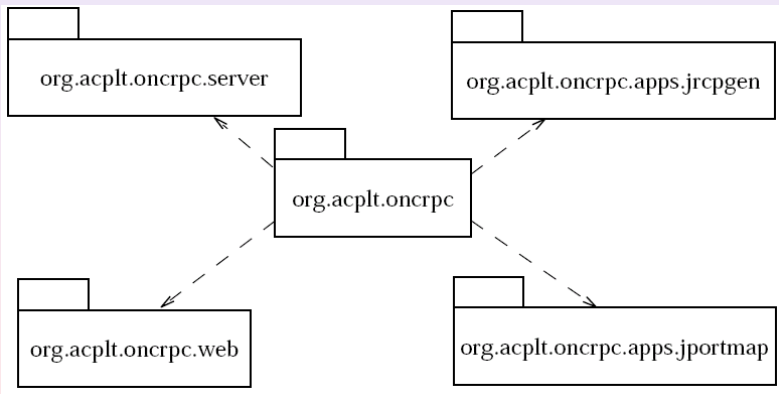
puis compilez et testez le programme !

- Pour obtenir la liste des services RPC enregistrés sur une machine :
 - `rpcinfo -p [machine]`
liste des services enregistrés
 - `rpcinfo -u machine num_prg [num_version]`
appel de la procédure 0 d'un programme en utilisant le protocole udp
 - `rpcinfo -t machine num_prg [num_version]`

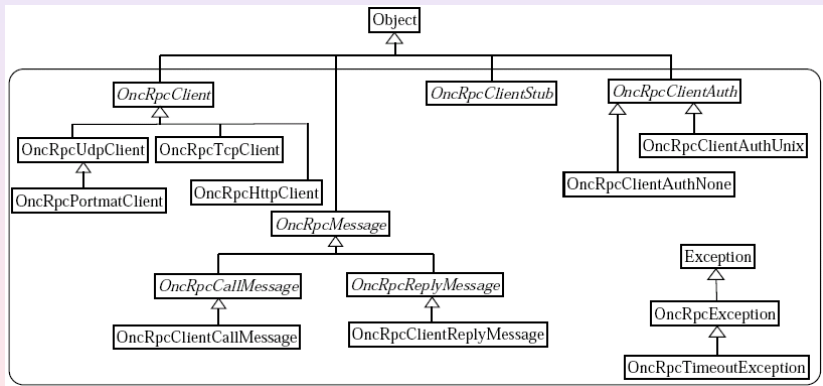
Les RPC en JAVA

- Par défaut Java ne propose pas de classes permettant de faire des clients ou des serveurs RPC compatible avec l'outil développé par Sun.
- Un ensemble de classes Open Source a été développé par ACPLT (Aachen Process Control Engineering) :
 - RemoteTea v1.0.4 (au 11 novembre 2004)
 - <http://acplt.org/ks/remotetea.html>
- Ce projet propose aussi :
 - Un compilateur RPCL → Java (jrpcgen).
 - Un "portmapper" écrit en Java (jportmap).

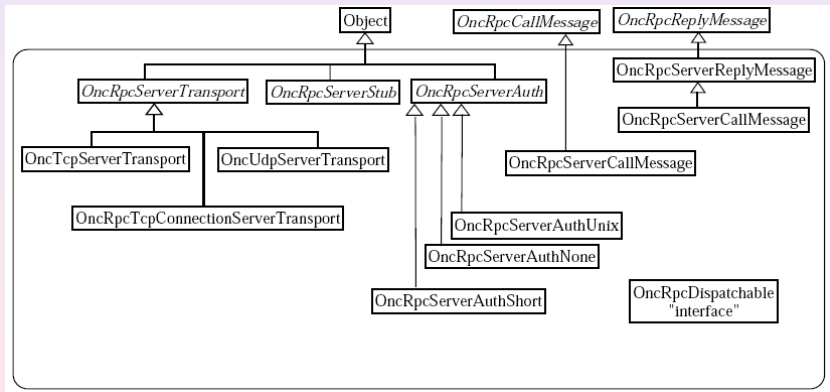
Les packages de Remote Tea



Le package org.acplt.oncrpc



Le package org.acplt.oncrpc.server



Fonctionnement général

- Le stub serveur (Sous-classe de `OncRpcServerStub`) :
 - S'enregistre au près du portmap (instanciation de l'attribut `info`).
 - Spécifie les flux XDR utilisables (instanciation de l'attribut `transport`) en précisant une méthode de dispatch (désignée par une instance d'une classe qui implémente `OncRpcDispatchable`).
- Le stub client (Sous-classe de `OncRpcClientStub`) :
 - Se connecte au serveur (en invoquant l'un des deux constructeurs de `OncRpcClientStub`). L'attribut client est alors instancié (objet de la classe `OncRpcClient`).
 - Invoque une procédure via la méthode `call` de client.

L'outil jrpcgen (1/2)

- C'est le compilateur qui permet de créer automatiquement les stubs (client et serveur) à partir d'un fichier en langage RPCL
- Utilisation :

```
java -jar jrpcgen [options] fichier.x
```
- Produit les fichiers suivants :
 - fichier.java : interface précisant les constantes
 - fichierClient.java : classe proposant l'appel des procédures distantes
 - fichierServerStub.java : classe abstraite à spécialiser pour implémenter les procédures distantes
 - xx.java : une classe par type déclaré dans fichier.x

L'outil jrpcgen (2/2)

- les options :

- c <NomClasse> spécifie le nom du stub client
- d <Repertoire> spécifie le répertoire destination
- p <NomPackage> spécifie le nom du package
- s <NomClasse> spécifie le nom du stub serveur
 - nobackup ne crée pas de sauvegarde des fichiers précédemment générés
 - noclient ne crée pas le stub client
 - noserver ne crée pas le stub serveur
- parseonly vérifie juste la syntaxe du .x

Un exemple : HelloWorld en JAVA (1/3)

- Le fichier de description HelloWorld.x :

```
typedef string Chaîne<255>;  
program HELLO_WORLD_PROG {  
    version HELLO_WORLD_VERSION_1 {  
        void hello_world_null(void)=0;  
        Chaîne hello_world(Chaîne)=1;  
    }=1;  
} = 0x22222220;
```

- Génération des souches :

```
java -jar jrpcgen.jar HelloWorld.x
```

- Compilation des programmes :

```
javac -classpath ./oncrpc.jar:./jrpcgen.jar:./jportmap.jar:. *.java
```

- Vous pouvez télécharger l'ensemble des sources du programme à l'adresse

<http://litis.univ-lehavre.fr/~duvallet/enseignements/enseignements-MATIS-fr.php>

Un exemple : HelloWorld en JAVA (2/3)

- Le programme HelloServer.java :

```
import org.acplt.oncrpc.*;
import org.acplt.oncrpc.server.*;
import java.io.IOException;
import java.net.InetAddress;

public class HelloServer extends HelloWorldServerStub{

    public HelloServer () throws OncRpcException, IOException {
        super ();
    }

    public void hello_world_null_1(){
        System.out.println ("Connexion d'un client");
    }

    public Chaine hello_world_1(Chaine arg1){
        System.out.println ("Connexion d'un client");
        System.out.println ("Reception de la chaine : "+arg1.value);
        return new Chaine ("Hello "+arg1.value+" !");
    }

    public static void main (String args []){
        try{
            new HelloServer().run ();
        }
        catch (Exception e){
```


Un exemple : HelloWorld en JAVA (3/3)

- Le programme HelloClient.java :

```
import org.acplt.oncrpc.*;
import java.io.IOException;
import java.net.InetAddress;

public class HelloClient {

    public static void main (String args []){
        try{
            HelloWorldClient h = new HelloWorldClient (InetAddress.getLocalHost(),
                                                       OncRpcProtocols.ONCRPC_UDP);
            Chaine result = h.hello_world_1 (new Chaine ("Claude"));
            System.out.println ("Resultat:"+result.value);
        }
        catch (Exception e){
            System.out.println ("Erreur:"+e);
        }
    }
}
```