Introducción a RPC

Juan Pavón Mestras
Dep. Sistemas Informáticos y Programación
Universidad Complutense de Madrid
jpavon@sip.ucm.es
http://bogart.sip.ucm.es/~juan

RPC

Llamada a procedimiento remoto
(*Remote Procedure Call*)

- Birrell y Nelson (1984)
  - Intentar que los programas puedan llamar a procedimientos localizados en otras máquinas
    - De manera similar a como se hace una llamada a procedimiento local
    - Proporciona transparencia de distribución
  - Cuando un proceso en una máquina A llama a un procedimiento en la máquina B:
    - El proceso que realiza la llamada desde A se suspende
    - La ejecución del procedimiento se realiza en B
    - La información se puede pasar de un proceso a otro como parámetros, y regresar como resultado del procedimiento
      - El programador no se preocupa de la transferencia de mensajes

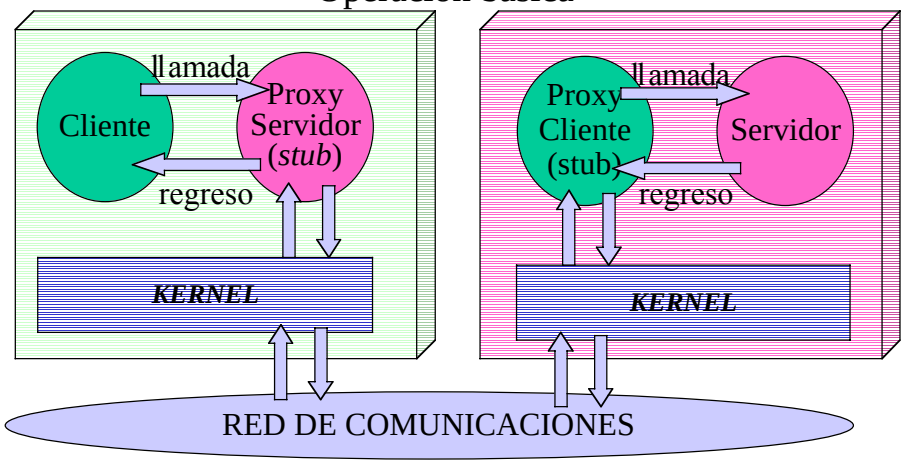
RPC

Llamada a procedimiento remoto
(*Remote Procedure Call*)

- Problemas que resuelve:
  - Ambos procesos están en espacios de direcciones distintos
  - Transferencia de parámetros y resultados
  - Heterogeneidad
    - Qué pasa si las dos máquinas tienen arquitecturas distintas
  - Fiabilidad
    - Qué pasa si hay fallos en alguna de las máquinas
    - Qué pasa si hay fallos en el canal de comunicaciones
  - Localización y selección de servicios
  - Seguridad

RPC

Operación básica



RPC

Automatización

- Ayuda a la transparencia de distribución
  - RPC esconde el código de red en los *stubs* de cliente y servidor
  - Los stubs se generan automáticamente

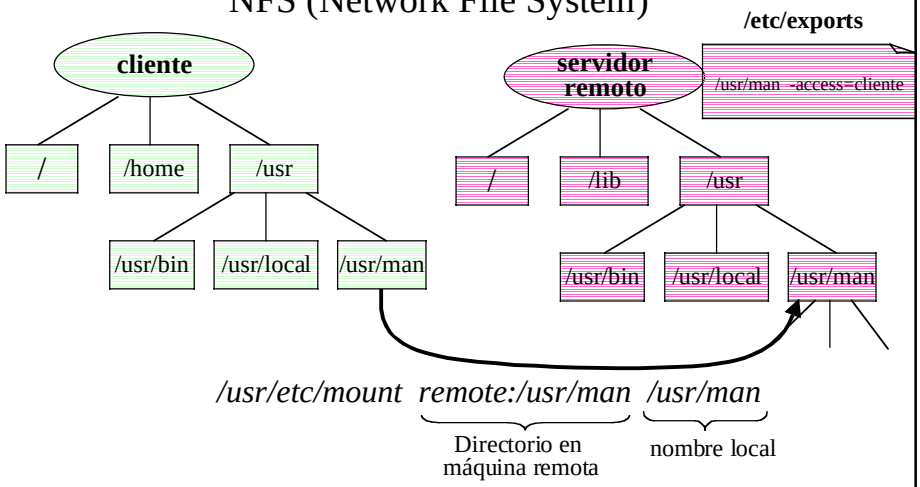
    rpcgen especificación
- Se protege a la aplicación de los detalles de red
  - Sockets, marshalling de los parámetros, orden de los bytes, temporizadores, control de flujo, reconocimientos, retransmisiones, etc.
- Permite seleccionar llamadas locales a procedimientos o remotas, según la localización del procedimiento llamado

Plataformas RPC

- OSF/DCE (Distributed Computing Environment)
  - Paquete de multithreading
  - Varios modelos de RPC y varios tipos de representación de datos y protocolos de red
  - Servicios: Tiempo, Directorio, Seguridad, Archivos
- ONC RPC
  - Antes Sun RPC/XDR
  - RFCs 1831 (ONC RPC) y 1832 (XDR)
- Extensiones a arquitecturas de computación de objetos distribuidos
  - Java RMI
  - OMG CORBA

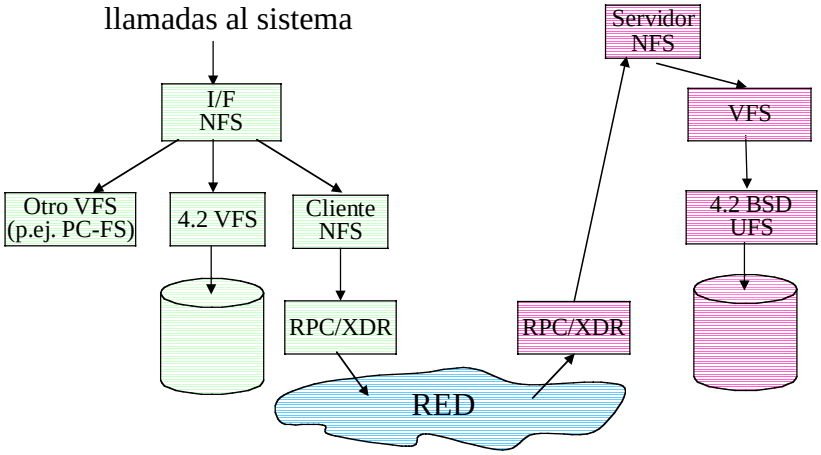
Servicios sobre RPC

NFS (Network File System)



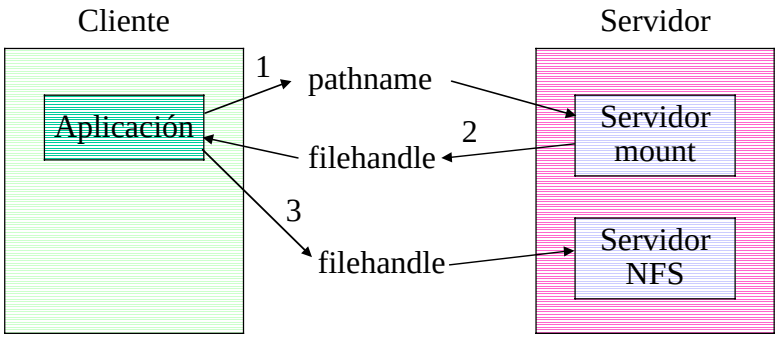
Servicios sobre RPC

NFS (Network File System)



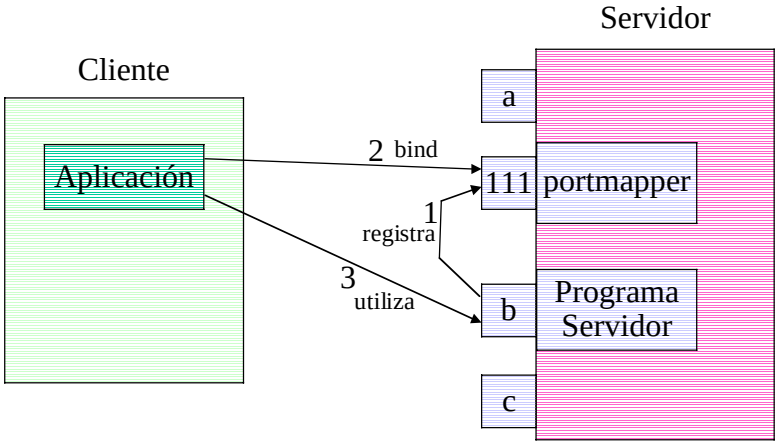
Servicios sobre RPC

NFS (Network File System)



Servicios sobre RPC

Port Mapper



Servicios sobre RPC

Especificación del protocolo del Port Mapper en RPC

```
const PMAP_PORT = 111;              /* portmapper port number */
/* A mapping of (program, version, protocol) to port number  */
struct mapping {
        unsigned int prog;
        unsigned int vers;
        unsigned int prot;
        unsigned int port;
};
Struct *pmaplist {
        mapping map;
        pmaplist next;
};
/* Port mapper procedures */
program PMAP_PROG {
        version PMAP_VERS {
                void PMAPPROC_NULL(void)                = 0;
                bool PMAPPROC_SET(mapping)              = 1;
                bool PMAPPROC_UNSET(mapping)            = 2;
                unsigned int PMAPPROC_GETPORT(mapping)  = 3;
                pmaplist PMAPPROC_DUMP(void)            = 4;
        } = 2;
} = 100000;
```


Servicios sobre RPC

NIS (Network Information Service)

☞ También conocido como "páginas amarillas" (*yellow pages*)

- Proporciona gestión centralizada de información de red (no sólo de máquinas y direcciones IP, también de passwords, servicios, grupos, aliases, etc.)
- La información se guarda en NIS maps
  - Conjuntos de claves y valores asociados
  - Implementados en archivos dbm localizados en /etc/yp en las máquinas de los servidores NIS
    - ☞ reemplazan o aumentan la información de Unix en /etc/…
  - Ejemplos de mapas NIS: hosts, protocols, password, rpc, services, group, netgroup, aliases, timezone, …
  - NIS domain: un conjunto de NIS maps
    - Define un área de control administrativo

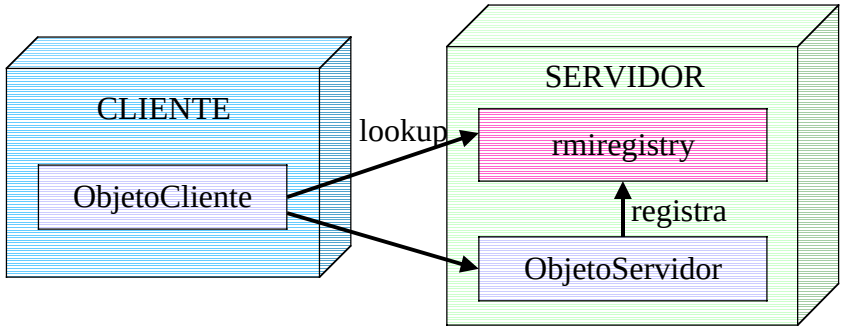
Servicios sobre RPC

NIS (Network Information Service)

- Para cada NIS map hay
  - un servidor NIS maestro
  - y los demás funcionan como esclavos
- Un servidor NIS se ejecuta como el demonio *ypserv*
  - Se puede invocar con la orden *ypinit*
- Los clientes NIS, en vez de acceder a los archivos /etc hacen RPC al servidor NIS cada vez que necesitan información de una base de datos NIS
  - *ypwhich* dice el nombre del servidor NIS que usa *ypbind*
  - *ypcat* passwd -- equivaldría a hacer cat /etc/passwd
  - *ypmatch* nombreusuario passwd -- ypcat passwd | grep nombreusuario

Java RMI

- RMI (*Remote Method Invocation*)
  - Permite acceder a métodos de objetos remotos



Programación con Java RMI

Definición de interfaz

- La interfaz define los servicios ofrecidos por el objeto remoto
- Ejemplo:

```
package Contador;
public interface ContadorRMI extends java.rmi.Remote {
    int getCuenta() throws java.rmi.RemoteException;
    void setCuenta(int valor) throws java.rmi.RemoteException;
}
```

  - La interfaz debe heredar de *java.rmi.Remote*
  - Las excepciones *java.rmi.RemoteException* se pueden producir por problemas de comunicación, o caída del servidor

Programación del cliente con Java RMI

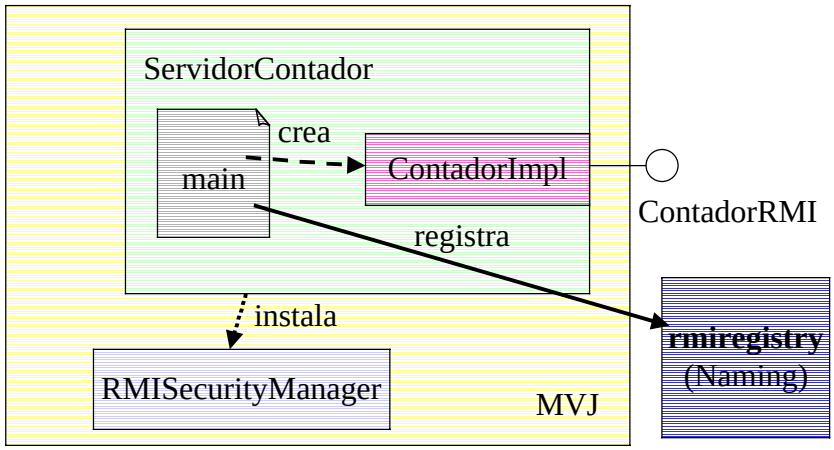
```
package Contador;
import java.io.DataInputStream;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class ClienteContadorRMI {

  public static void main(String[] args) {
    DataInputStream stdin =
            new DataInputStream(System.in);
    System.setSecurityManager(
            new RMISecurityManager());
    try {
      ContadorRMI stubContador =
          (ContadorRMI) Naming.lookup("rmi://" +
            args[0] + "/primer contador");
      System.out.println(
            "Elija una opción: salir | ver | numero");
      while (true) {
            String entrada = stdin.readLine();
            if ( entrada.equals("salir") ) break;
            if ( entrada.equals("ver") )
              System.out.println(
                        stubContador.getCuenta());
            else {
              int intVal = Integer.parseInt(entrada);
              stubContador.setCuenta(intVal);
            }
          }
          System.out.println("Adios.");
      }
      catch(Exception e) {
        e.printStackTrace();
      }
    }
}
```

Programación del servidor con Java RMI

Estructura del servidor



Programación del servidor con Java RMI

```
package Contador;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ContadorImpl extends UnicastRemoteObject implements ContadorRMI {
   private int cuenta;
   public ContadorImpl() throws RemoteException {
     super();
     System.out.println(”Objeto ContadorImpl: constructor");
     cuenta = 0;
   }
   public synchronized int getCuenta() throws RemoteException {
     System.out.println(" Objeto ContadorImpl: getCuenta() ");
     return cuenta;
   }
   public synchronized void setCuenta(int valor) throws RemoteException {
     System.out.println(" Objeto ContadorImpl: setCuenta (”+valor+”)”);
     cuenta = valor;
   }
}
```

Programación del servidor con Java RMI

```
package Contador;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class ServidorContadorRMI {  // crea el objeto ContadorImpl
   public static void main(String[] args) {
      System.setSecurityManager(new RMISecurityManager());
      try {
         ContadorImpl miContador = new ContadorImpl();
         Naming.rebind(”primer contador", miContador);
      } catch(Exception e) {
         System.out.println("Exception: " + e.getMessage());
         e.printStackTrace();
      }
   }
}
```


Ejecución de aplicación con Java RMI

1) Compilar el servidor
   - *javac ServidorContador.java*
     - El compilador Java se encarga de compilar *ContadorRMI* y *ContadorImpl*, por estar usados en *ServidorContador*.
2) Creación de stubs:
   - *rmic Contador.ContadorImpl*
     - Crea las clases *ContadorImpl_Stub* y *ContadorImpl_Skel*
3) Compilar el cliente:
   - *javac Contador.ClienteContador*
4) Arrancar el RMI Registry:
   - *start rmiregistry*
5) Arrancar el servidor:
   - *java Contador.ServidorContador*
6) Ejecutar el cliente:
   - *java Contador.ClienteContador maquinaservidor*

¡¡¡ Comprobad la variable CLASSPATH !!!

Bibliografía

- Java RMI:
  - http://java.sun.com/products/jdk/rmi (páginas oficiales de Sun)
  - http://www.parallax.co.uk/cetus/oo_javabeans.html (Cetus links)
- RPC
  - A. Birrell y B. Nelson, *Implementing Remote Procedure Calls*, ACM Transactions on Programming Languages and Systems, vol.2, n. 1, feb. 1984, pp. 39-59
  - A. Birrell, *Secure Communication Using Remote Procedure Calls*, ACM Transactions on Computer Systems, vol. 3, n. 1, Feb 85, pp. 1-14
  - OSF, *Introduction to OSF DCE*, Englewood Cliffs, Prentice Hall, 1992