Sistemas Distribuidos

Llamados a Procedimientos Remotos

Arturo Díaz Pérez

Sección de Computación
Departamento de Ingeniería Eléctrica
CINVESTAV-IPN
Av. Instituto Politécnico Nacional No. 2508
Col. San Pedro Zacatenco
México, D. F. CP 07300

Tel. (5)747 3800 Ext. 3352
e-mail: adiaz@cs.cinvestav.mx

Introducción

- Las primitivas de comunicación afectan el diseño e implantación de un sistema distribuido
  - Muchos conjuntos de primitivas son funcionalmente equivalentes pero la eficiencia y el modelo conceptual pueden ser diferentes
- El modelo de intercambio de mensajes se basa en dos primitivas
  - SEND(dest, msg)
  - RECEIVE(source, msg)
    - Bloqueantes: el emisor se bloquea hasta que el mensaje es enviado o recibido y el receptor se bloquea hasta que se recibe el mensaje
    - No bloqueantes: ni emisores ni receptores se bloquean. El envío puede ocurrir de manera concurrente.
    - Síncrona: los pares de la comunicación siempre se bloquean.
    - Asíncrona: se almacena el mensaje en un buffer intermedio, permitiendo implantaciones bloqueantes o no bloqueantes.
- ¿Cómo se construye un sistema distribuido?
  - Ignorar detalles de la implantación como protocolo de comunicación, medio de comunicación y direccionamiento
  - Usando un modelo muy popular: *cliente-servidor*

Modelo Cliente Servidor

- Conceptos de Alto Nivel
  - El sistema completo se descompone en pares: cliente-servidor:
    - ***Servidores*** (procesos u objetos) son especializados, pocos, y proporcionan ***servicios***
      - ✓ exporta una interfaz
    - ***Clientes*** (procesos u objetos) son solicitantes de servicios (o consumidores).
      - ✓ invoca la interfaz
    - Un servidor actúa como una instancia de un ***tipo de datos abstracto***.
- Algunas características del modelo Cliente/Servidor son:
  - Un cliente solicita un servicio, un servidor lo proveé
  - La relación cliente-servidor es por interfaz
    - A puede ser servidor y B puede ser cliente de un servicio X
    - B puede ser servidor y A puede ser cliente de un servicio Y
      - ✓ ***la relación no es por nodo o por proceso***
    - En la práctica la mayoría de los servidores son dedicados
    - La relación cliente-servidor puede ser anidada
      - ✓ A invoca al servidor B, B invoca al servidor C, ...

Sistemas Distribuidos *RPC*-3

Modelos Alternativos

- Memoria Compartida Distribuida
- Modelo de Objetos Distribuidos
- Sin embargo, la organización cliente-servidor es:
  - intuitiva, conveniente, común (natural)
  - el modo dominante para construir sistemas distribuidos
    - puede cambiar en el futuro con técnicas orientadas a objetos
      - ✓ los modelos orientados a objetos pueden ocultar el modelo cliente-servidor o pueden encapsularlo mejor, pero tendrán que demostrarlo.
- ¿Cómo construir aplicaciones cliente-servidor?
  - Intercambio asíncrono de mensajes
    - el emisor puede continuar trabajando mientras el receptor procesa la respuesta a su solicitud
  - Problemas:
    - ¡el emisor realmente no tiene gran cosa que realizar !
    - la programación es relativamente complicada
      - ✓ el procesamiento de solicitudes múltiples requiere un empatar solicitudes con respuestas
      - ✓ fallas potenciales
    - se puede explotar el paralelismo teniendo varios hilos (threads) de ejecución en el cliente

Sistemas Distribuidos *RPC*-4

Llamados a Procedimientos Remotos

- RPC
  - Cualquier interfaz o servicio *local* puede ser potencialmente una interfaz *remota*
    - **Impresora**: enfila un archivo, retira un archivo, inicia la cola, vacía la cola, etc.
    - **Sistema de archivos**: Crear, borrar, leer y escribir archivos
    - **Manejador de candados**: Obtener un candado de lectura o escritura, liberar un candado, etc.
    - **Servidor de tiempo**: Iniciar el temporizador, obtener el valor del temporizador, etc.
    - **Servidor de nombres**: registrar un nombre, retirar el registro de un nombre, consultar un nombre
  - Es necesario una abstracción de la comunicación para proporcionar servicios remotos
  - Simple: análogo a los llamados locales a procedimientos o llamados al sistema
    - hacer un llamado a una función, y
    - dejar a la función misma preocuparse por los detalles de la comunicación
  - Conveniente: los programadores saben cómo hacer llamados a funciones
    - facilita la construcción de sistemas distribuidos
      - ✓ ¿es razonable esperar una evolución similar en métodos de invocación de objetos?
  - Los clientes llaman a los servidores (manejadores de recursos)
    - Los servidores tienen un nombre o identificación único.

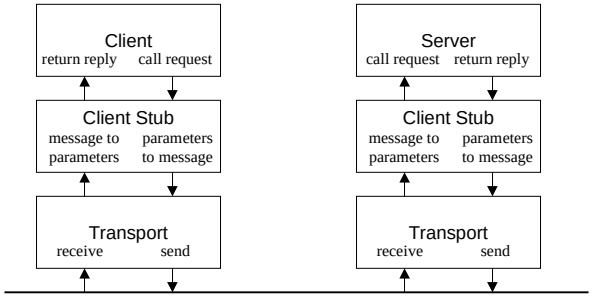
Sistemas Distribuidos *RPC*-5

Aspectos Principales de los RPCs

- Control de flujo: transferencia síncrona del control
  - el que invoca hace una solicitud y se bloquea
  - el invocado sirve la solicitud y contesta
  - el que invoca continua su flujo de ejecución
- Sintaxis de la invocación
  - RPCs tienen parámetros de entrada y salida
  - No hay información de variables globales
  - El paso de apuntadores no tiene sentido
  - Las estructuras de datos deben ser “aplanadas”
- Tipos de RPCs
  - Codificados en un lenguaje de programación especial
    - Ada distribuido, Cedar, Argus, Arjuna, *Java*
  - Codificados en un lenguaje especial para definición de interfaces
    - DCE en OSF, IDL en Corba, RPCGEN de Sun
    - La definición de interfaces consiste de una lista de ***firmas*** (prototipos)**.**
      - ✓ nombres de procedimientos,
      - ✓ argumentos de entrada y de salida

Sistemas Distribuidos *RPC*-6

Flujo de RPC



- Cliente
  - Debe haber un enlace entre el lenguaje de programación y el sistema RPC
  - Genera un "stub" que represena el RPC en el cliente
  - El procedimiento stub crea un mensaje empacado con los valores de los parámetros
  - Bloquea
- Servidor
  - Los mecanismos de RPC proporcionan un despachador
  - El despachador llama al "stub" del servidor
  - El procedimiento stub desempaca los parámetros del cliente y llama a un procedimiento local
  - Regresa los argumentos de regreso siguiendo un procedimiento similar

Sistemas Distribuidos *RPC-7*

Problemas en la Operación de RPCs

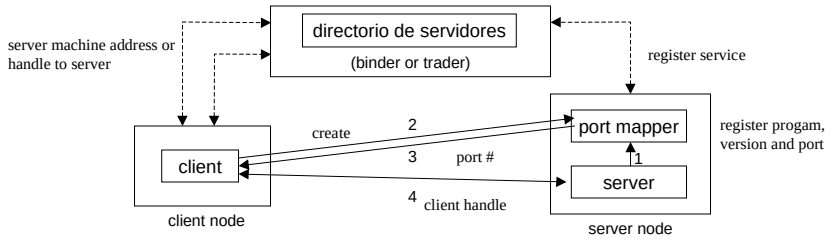
- Paso de parámetros y conversión de datos
  - ¿Cuáles tipos de datps pueden ser pasados y cómo se representan los datos en mensajes?
- Binding
  - ¿Cómo un cliente localiza a un servidor y cómo un servidor registra sus servicios (haciéndolos remotamente visibles)?
- Compilación
  - ¿Cómo se producen los procedimientos "stub" y cómo se ligan a los procesos clientes y servidores?
- Manejo de fallas y excepciones
  - ¿Cómo se reportan los errores y qué suposiciones se pueden hacer acerca de las fallas en el sistema?
- Seguridad
  - ¿Qué se puede decir acerca de RPCs seguros?

Sistemas Distribuidos *RPC-8*

Paso de Parámetros

- El manejo de parámetros incluye el paso y la conversión de datos en mensajes: responsabilidad de los procedimientos stub.
  - Llamado por valor
    - Se copia a una variable local al entrar a un procedimiento. No hay modificaciones
  - Llamado por nombre
    - Evaluación en tiempo de ejecución de expresiones simbólicas. No adecuado para RPCs
  - Llamado por referencia
    - No hay un espacio de direccionamiento compartido, por lo que no tiene sentido el uso de apuntadores.
  - Llamado por copia y restauración
    - Combinación del llamado por valor a la entrada del procedimiento y el llamado por referencia a la salida del procedimiento. Adecuado para RPCs
- La verificación de tipos de datos
- Representación de datos
- Sintaxis de transferencia de datos
  - ASN.1 (Abstract Syntax Notation One) es in lenguaje que puede ser usado para definir estructuras de datos
    - Se ha usado para especificar formatos de unidades de datos de protocolos en redes.

Binding



- Binding entre clientes y servidores
  - Cuando inicia un servidor, se registra enviando una solicitud al "port mapper" indicando número de programa, de versión y de puerto que el servidor atenderá.
  - Antes de hacer llamados remotos, un cliente debe contactar al "port mapper" para obtener un identificador ("handle") para accesar a un servidor. Se indica número de programa y de versión
  - El "port mapper" regresa el número de puerto que el servidor registró
  - El sistema cliente construye un identificador del cliente para el proceso elcual será usado en llamados remotos subsecuentes.
  - En un caso más general, el contacto entre clientes y servidores es mediante un "binder"

Lenguaje de Definición de Interfaces

- Los servicios se especifican mediante un lenguaje de definición de interfaces (IDL)
  - XDR: eXternal Data Representation

```
program PROGRAM-NAME {
    version VERSION-NAME {
        long PROCEDURE_A( parameters ) = 1;      /* procedure number 1*/
        string PROCEDURE_B( parameters ) = 2;    /* procedure number 2*/
    } = 1;                                       /* version number 1*/
} =  12345;                                      /* program number 12345 */
```

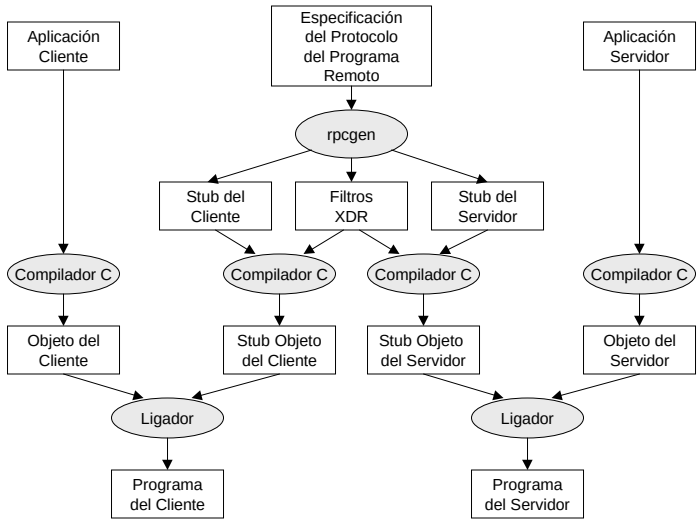
  - Es necesario un programa en C, por ejemplo, para el cliente
    - Incluye los llamados a los procedimientos
  - Es necesario un programa en C, por ejemplo, para el servidor
    - Incluye la codificación de los procedimientos

```
#include "rdb.h"
...
main( ...)
{
    ...
    PROCEDURE_A( .... );
    ...
    PROCEDURE_B( ... );
}
```

```
#include "rdb.h"
...
long PROCEDURE_A( .... )
{    ...  }
    . . .
string PROCEDURE_B( ... )
{    ...  }
}
```

Sistemas Distribuidos *RPC*-11

Compilación



Sistemas Distribuidos *RPC*-12

Manejo de Excepciones y Fallas

- Manejo de excepciones
  - ¿Cómo un servidor reporta la información de un estado a un cliente?
  - ¿Cómo un cliente envía información de control a un cliente?
  - Procedimientos locales: variables globales compartidas y señales (*errno*)
  - El intercambio de información de estado y control se debe hacer en un canal de datos
    - Se puede usar un canal separado
    - Normalmente se implanta como parte de una bliblioteca de soporte de "stubs" y es transparente a los procesos cliente.
- Manejo de fallas
  - Debido al incremento de fallas en los nodos, los RPCs son mucho más probables de fallar que los procedimientos ordinarios
  - Fallas de clientes y servidores es independiente
    - Falla en la red
    - Caídas en el nodo del servidor o en el proceso del servidor
    - Comunicación no transitiva, (a -> b, b -> c, pero no c -> a)
  - Retransmitir hasta que se obtenga una respuesta
  - Usar idempotencia, invocaciones repetidos no tienen efecto

Semánticas de Llamado

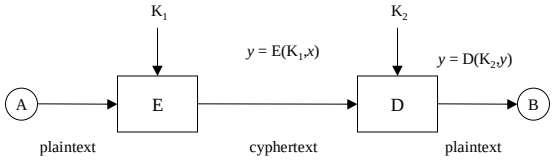
- Existen las siguientes semánticas de llamado en RPCs
  - "*Tal vez*"
    - no existen medidas de tolerancia a falla
    - si no hay respuestas, no se sabe si el procedimiento remoto ha sido ejecutado, el cliente retransmite su solicitud
  - "*Al menos una vez*"
    - El servidor manda una excepción y el cliente reintenta la operación cuando el servidor se recupera
    - Se retransmite hasta obtener una respuesta
    - Adecuado para operaciones idempotentes
  - "*A lo más una vez*"
    - El servidor manda una excepción y el cliente renuncia a la operación inmediatamente
    - Se retransmiten mensajes filtrados

Seguridad

- Aspectos
  - RPC es una forma de ejecución remota que permite a los programas o comandos ejecutarse en otros sistemas
    - RPC son vulnerables a ataques de usuarios remotos
  - RPC son el mecanismo básico para construir aplicaciones cliente-servidor
    - Los rasgos de seguridad deben construirse sobre RPCs seguros
  - Autenticación mutua
    - Se verifican las identidades de clientes y servidores
    - La autenticidad se debe asegurar en mensajes y procesos que se comunican
  - Integridad, confidencialidad y originalidad.
    - Los mensajes deben conservarse íntegros
    - Su contenido no debe ser revelado (confidencialidad).
    - El mismo mensaje no debe aparecer más de una vez (originalidad).
  - Protocolo de autenticación
    - Depende de: los niveles de seguridad que se requieren, los tipos de ataques a que es sometido el sistema y de algunas limitaciones inherentes en el sistema.

Sistemas Distribuidos *RPC*-15

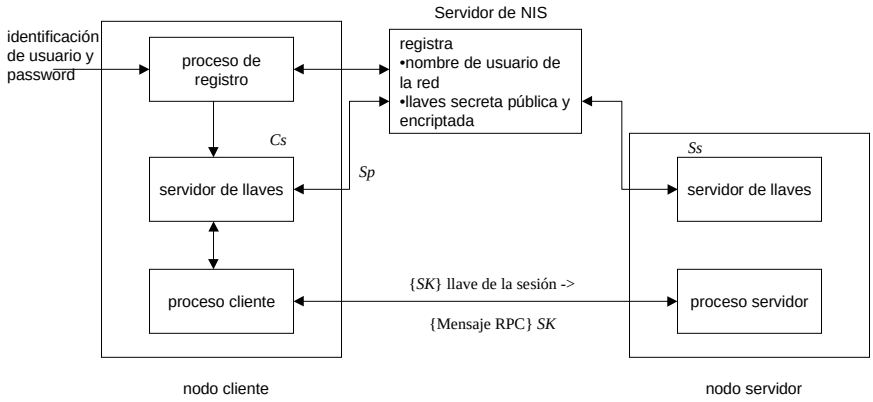
Encriptamiento



A	Público	B
	a, p	
x		y
a^x		a^y
	$a^x, a^y \bmod p$	
a^y		a^x
$(a^y)^x$		$(a^x)^y$
$k = a^{xy}$		$k = a^{xy}$

Sistemas Distribuidos *RPC*-16

Seguridad en RPC



Sistemas Distribuidos *RPC*-17

Intercambio de Llaves Exponenciales

- El proceso de registro deposita la llave secreta del cliente, *Cs*, en la memoria del servidor de llaves
- Similarmente, cuando el servidor arranca, la llave secreta del servidor, *Ss*, es depositada en la memoria del servidor de llaves
  - Los procesos servidores de llaves son responsable de generar una llave para la sesión
  - Se asignan un par de llaves, públicas y secretas, se asignan para cada cliente y cada servidor, (C_s,C_p) y (S_s,S_p).
    - Cs y Ss son números aleatorios de 128 bits
    - $C_p = \alpha^{Cs} \bmod M$ y $S_p = \alpha^{Ss} \bmod M$, donde α y M son constantes conocidas.
    - En el lado del cliente: $SK_{cs} = S_p^{Ss} = (\alpha^{Ss})^{Cs} = \alpha^{Ss*Cs}$
    - En el lado del servidor: $SK_{sc} = C_p^{Ss} = (\alpha^{Cs})^{Ss} = \alpha^{Cs*Ss}$
    - Se puede ver que $SK_{cs} = SK_{sc} = SK$
- Cada mensaje de RPC se autentica mediante una llave de conversión *CK* de 56 bits de longitud generada aleatoriamente y transportada al servidor usando la llave de la sesión.
  - Estampa de tiempo, número de secuencia y código de recuperación de errores.

Sistemas Distribuidos *RPC*-18

Ejemplo: Definición XDR

```
/* FileReadWrite service interface definition
   in file FileReadWrite.x */
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Lenght;
struct Data {
    int  Length;
    char buffer[MAX];
    };
struct readargs {
    FileIdentifier  f;
    FilePointer    position;
    Length         length;
    };
```

```
program FILEREADWRITE {
   version VERSION {
      void WRITE( writeargs ) = 1;
      Data READ( readargs ) = 2;
      } = 2;
   } = 9999;
```


Ejemplo: Código para el Cliente y Servidor

```
/* File C.c  Simple client of the FileReadWrite Service */
#include <stdio.h>
#include <rpc/rpc.h>
#include "FileReadWrite.h"
main( int argc, char ** argv )
{
    CLIENT   *clientHandle;
    char     *serverName = "coffee";
    readargs  a;
    Data     *data;

    /* Creates socket and a client handle */
    clientHandle = clnt_create( serverName,
        FILEREADWRITE,  VERSION, "udp" );
    if( clientHandle == NULL ) {
         /* unable to contact server */
        clnt_pcreateerror( serverName );
        exit( 1 );
    }
    a.f = 10;
    a.position = 100;
    a.length = 1000;
     /* Call to remote read procedure */
    data = read_2( &a, clientHandle );
    ...
    clnt_destroy( clientHandle );    /* closes socket */
}
```

```
/* File S.c  Server Procedures of the FileReadWrite
    Service */
#include <stdio.h>
#include <rpc/rpc.h>
#include "FileReadWrite.h"

voide *write_2 ( writeargs *a )
{
    /* do the writing to the file   */
}

Data *read_2( readargs *a )
{
    static Data result;    /* must be static */
    result.buffer = ...    /* do the reading from the file */
    result.length = ...    /* amount read from the file */

    return &result;
}
```