Llamadas a Procedimientos Remotos (RPC)

INTRODUCCIÓN

Sistema de Procesamiento Distribuido:

Conjunto de computadoras conectadas en red que le dan la sensación al usuario de ser una sola computadora

Ventajas:

- Compartición De Recursos
- Concurrencia
- Alta escalabilidad
- Tolerancia a fallos

Desventajas:

- Mayor complejidad
- Aspectos de seguridad

Modo de Operación

Creación servidor ⇒ añadir stub de cliente y stub servidor

Clientes usan stub para llamar a los servicios ofrecidos por servidor

- No envían mensajes a través de las primitivas de comunicación.
- Realizan llamada a procedimiento stub con parámetros adecuados.
- Stub empaqueta parámetros, envía *send* al núcleo y llama a *receive*.
- Núcleo envía mensaje a núcleo remoto.

Máquina servidor→ Stub servidor llama a *receive* y espera mensaje

- Mensaje llega a núcleo remoto ⇒ lo deja en el buzón del stub que desempaqueta parámetros y realiza llamada al servidor.
- Stub servidor recibe resultados ⇒ los empaqueta para formato *send* y llama a *send* para enviárselos al cliente.
- Nuevo ciclo con *receive* a la espera de una nueva petición.

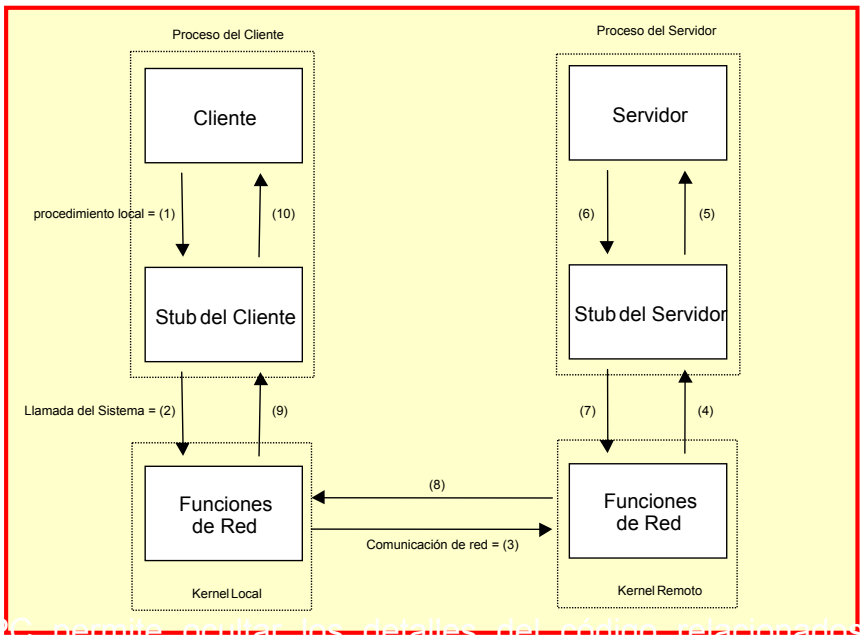
Mensaje resultados llega al núcleo cliente → buzón correspondiente

- Cliente llamó a *receive* ⇒ recoge los resultados del buffer.
- Stub desempaqueta los parámetros y devuelve el resultado.

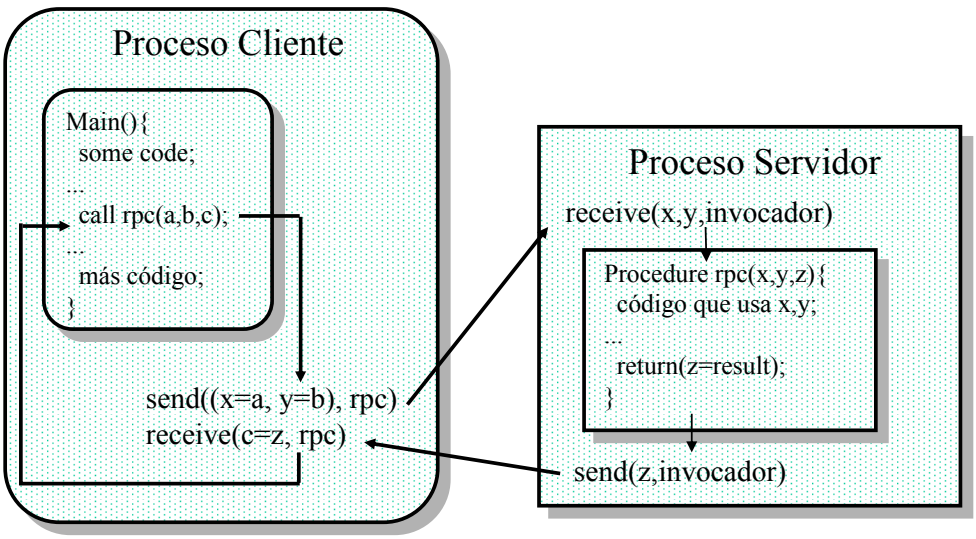
Llamadas a Procedimientos Remotos

- Creado por Bireel & Nelson en 1984.
- Permiten a los programas llamar procedimientos localizados en otras máquinas.
- Un proceso X en una máquina A, puede llamar un procedimiento localizado en una máquina B.
- Información puede llevarse del proceso invocador al invocado dentro de los parámetros.
- Ningún mensaje u operación de E/S es visible para el programador.
- Problemas a resolver:
  - Procedimiento invocador e invocado se ejecutan en diferentes máquinas, i.e. diferentes direcciones y posiblemente diferentes arquitecturas.
  - Ambas máquinas pueden fallar.

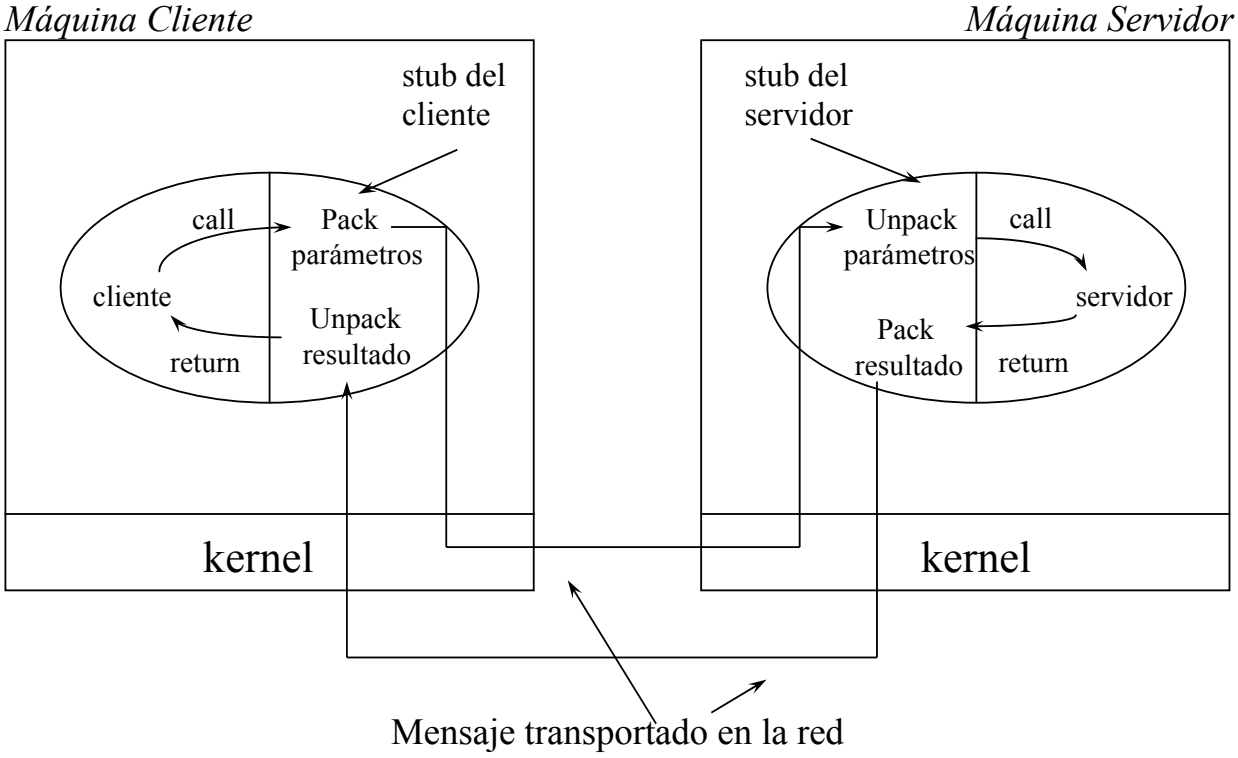
MODELO RPC



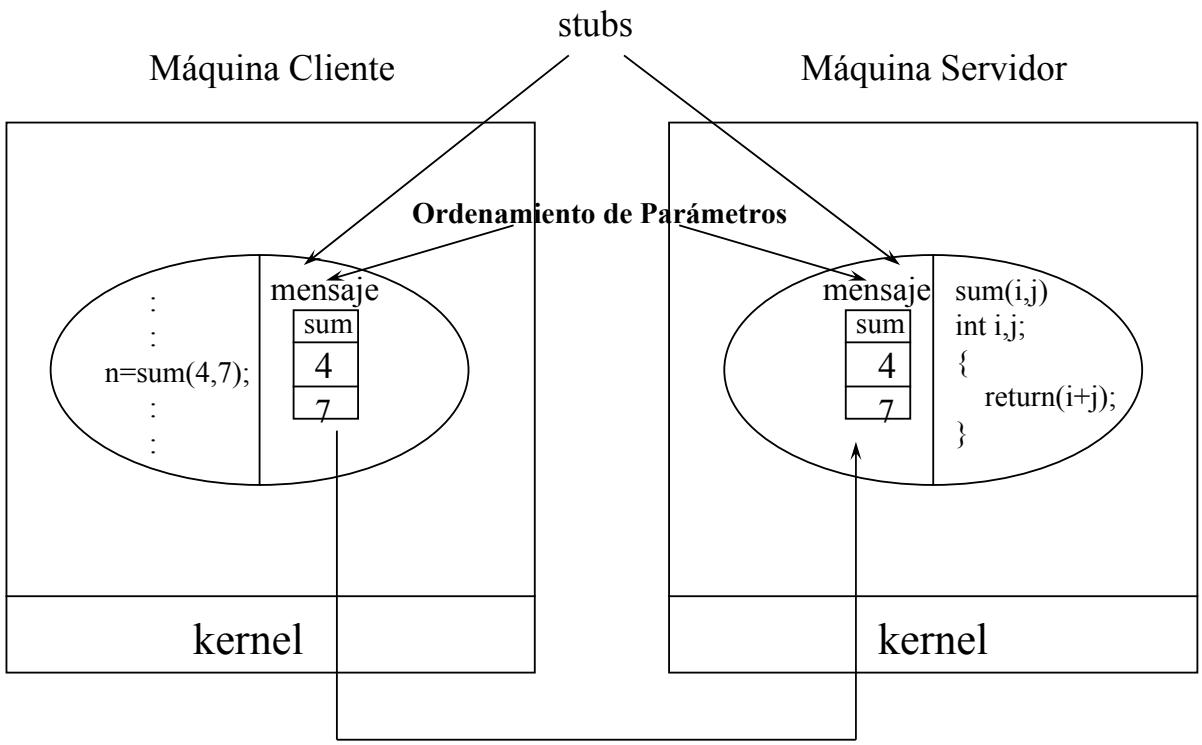
Llamadas a Procedimientos Remotos (RPC)



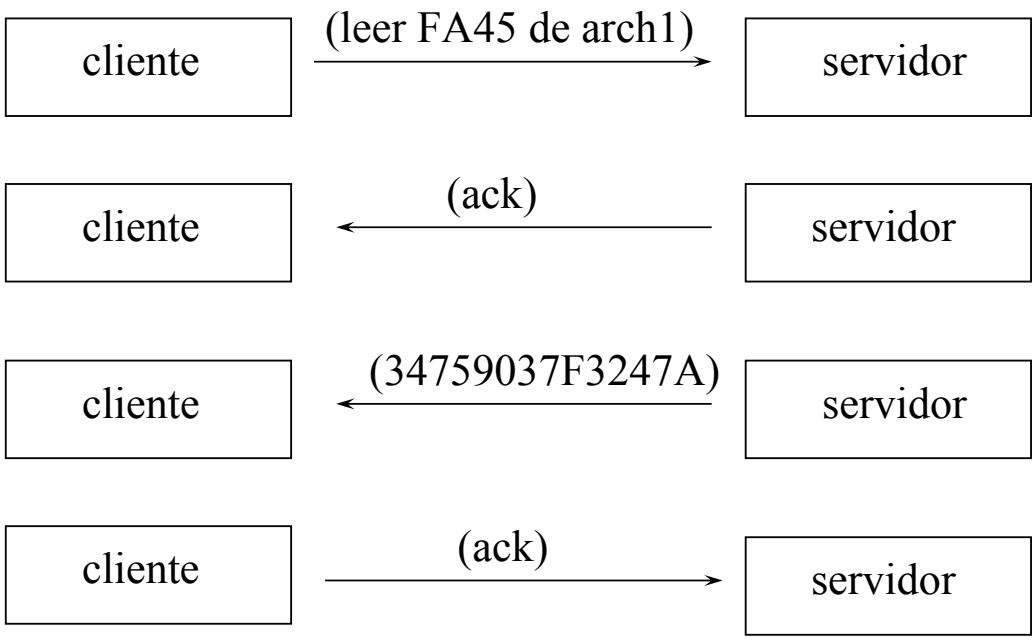
Modo de Operación



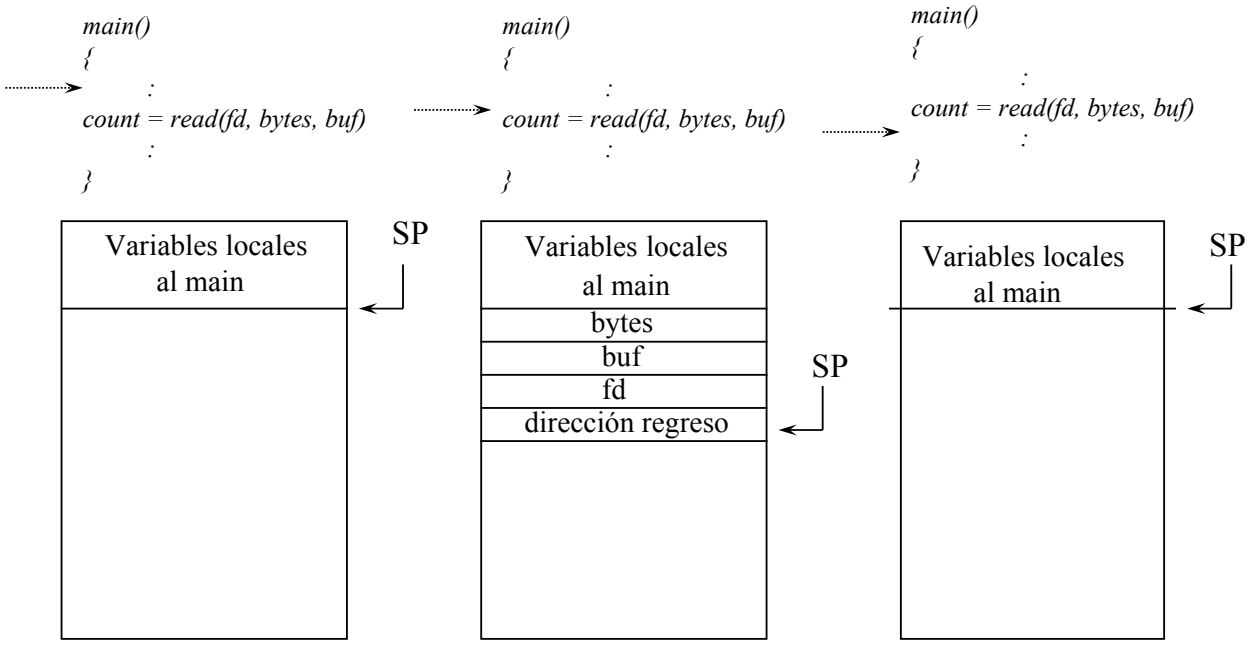
Transferencia de Parámetros en RPC



Modelo Cliente-Servidor, mensajes.



Ejecución de una Llamada Local



a) Stack antes llamada read

b) Stack durante ejecución read

c) Stack después llamada read

Tipos de Paso de Parámetros

- Por valor
  - en el stack se copia el valor del parámetro
  - valor de salida es igual al valor de entrada
- Por referencia
  - en el stack se almacena la dirección de la variable
  - es posible modificar el valor del parámetro
- Llamado con copia/restauración
  - se copia el valor de la variable en el stack, (como en paso por valor)
  - al final de la ejecución local se copia el valor que tiene la variable dentro del procedimiento, en el stack
  - el procedimiento que mandó llamar al procedimiento remoto copia el valor final en condiciones normales tiene el mismo efecto que el paso por referencia

Aspectos a Considerar en la Transferencia de Parámetros

- Tipos de paso de parámetros
  - Por valor.
  - Por referencia.
  - Llamada con copia/restauración.
- Problemas a resolver
  - Diferencias entre representación de datos (EBCDIC, ASCII), (Complementos a 1 y Complementos a 2).
  - Diferencias en formatos (Big Endian, Little Endian).
  - Paso de apuntadores, (estructuras complejas).
- Optimización
  - Especificación parámetros de entrada y salida.

Posibles Fallas en RPC

1. El cliente es incapaz de localizar al servidor.

2. El mensaje de petición del cliente al servidor se perdió.

3. El mensaje de respuesta del servidor al cliente se perdió.

4. El servidor falló (crashes) después de recibir una petición.

5. El cliente falló (crashes) después de enviar una petición.

RPCs: IMPLEMENTACIONES MÁS POPULARES

ONC-RCP
(Open Network Computing, ONC-RCP), desarrollada por Sun Microsystem y distribuida con casi todos los sistemas UNIX.

DCE-RPC
(DCE, Distributed Computing Enviroment) definido por la Fundación de Software Abierto (OSF, Open Software Foundation) e incluida en los sistemas operativos Windows.

CARACTERÍSTICAS DE TRANSPARENCIA

Pase de Parámetros

Los parámetros son pasados por valor. En cada procedimiento remoto se definen los argumentos de entrada y los valores de retorno

Enlace

El cliente contacta al sistema remoto apropiado para la ejecución de un procedimiento específico. Diferentes técnicas son empleadas para lograr este objetivo

Protocolo de Transporte

Soportan uno o dos protocolos. Específicamente, las implementación ONC y DCE soportan TCP y UDP como protocolos de transporte

CARACTERÍSTICAS DE TRANSPARENCIA

Manejo de Excepciones

En un ambiente de computación distribuida la posibilidad de fallo aumenta. La implementación debe proveer mecanismos para manejar tales situaciones.

Semántica de Llamadas

Existe tres tipos de semánticas de RPC:

- Exactamente una vez,
- Cuando mucho una vez, y
- Al menos una vez

CARACTERÍSTICAS DE TRANSPARENCIA

Representación de los Datos

Todas las implementaciones definen uno o más formatos estándar para los tipos de datos soportados por la implementación.

Desempeño

Usualmente el desempeño empleando RPC disminuye entre 10 y 100 veces en comparación con una llamada a un procedimiento local.

Seguridad

Con RPC existen los mismos problemas de seguridad relacionados con la ejecución remota de comandos.

ONC - RPC

- Desarrollada inicialmente por Sun Microsystem
- Disponible en la gran mayoría de los sistemas UNIX
- Especificación de ONC-RPC versión 2: RFC 1831
- Especificación de XDR: RFC 1832

ONC - RPC

ONC-RPC cuenta con los siguientes componentes:

- **rpcgen**: un compilador que toma la definición de la interfaz de un procedimiento remoto y genera el “stub” del cliente y el “stub” del servidor
- **XDR** (eXternal Data Representation): un estándar para la descripción y codificación de datos que garantiza portabilidad entre sistemas de arquitecturas diferentes
- Una librería que maneja todos los detalles

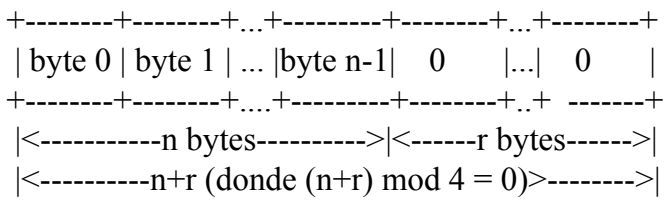
XDR (eXternal Data Representation)

- XDR es un protocolo estándar para la descripción y codificación de datos
- Útil para transferir datos entre diferentes arquitecturas computacionales
- Encaja dentro de la capa de presentación del modelo ISO
- Utiliza “Implicit Typing” (sólo viaja el valor de la variable por la red)
- Utiliza un lenguaje (similar a C) para describir los formatos de los datos. No es un lenguaje de programación
- RPC lo utiliza y extiende para **describir su formato** de datos y declarar procedimientos remotos
- Se asume como unidad fundamental de información el byte (= 8 bits) y que es portable

XDR (eXternal Data Representation)

Tamaño del Bloque

La representación de todos los items es en múltiplo de cuatro bytes (si es necesario se rellena con “ceros”para cumplir con esta condición)



XDR (eXternal Data Representation)

Algunos Tipos de Datos

Entero con signo

Rango: [-2147483648, 2147483647] (32 bits)
Representación: Complemento a 2
Declaración: `int identifier;`

Entero sin signo

Rango: [0, 4294967295] (32 bits)
Declaración: `unsigned int identifier;`

Enteros de 64 Bits

Declaración: `hyper identifier;`
`unsigned hyper identifier;`

XDR (eXternal Data Representation)

Algunos Tipos de Datos

Enumeración

Tiene la misma representación de los enteros con signo.
Declaración:

```
enum {name-identifier=constant,...} identifier;
```

Ejemplo:

```
enum {RED=2,YELLOW=3,BLUE=5} colors;
```

XDR (eXternal Data Representation)

Algunos Tipos de Datos

Punto Flotante de Presición Simple

Codificación: IEEE 754 (32 bits)
Declaración: `float identifier;`

Punto Flotante de Doble Presición

Codificación: IEEE 754 (64 bits)
Declaración: `double identifier;`

Punto Flotante de Cuádruple Presición

Codificación: una extensión al estándar IEEE 754 (128 bits)
Declaración: `quadruple identifier;`

XDR (eXternal Data Representation)

Algunos Tipos de Datos

Data Opaca

Data sin interpretación.
Declaración:

```
opaque identifier[n]; (de longitud fija)
opaque identifier <m>; (de longitud variable)
```

Cadenas de Caracteres

Declaración: `string object<m>;`

XDR (eXternal Data Representation)

Algunos Tipos de Datos

Arreglo

Declaración:

```
type-name identifier[n]; (Arreglo de longitud fija)
type-name identifier<m>; (Arreglo de longitud variable)
```

Estructura

Declaración:

```
struct {
   component-declaration-A;
   component-declaration-B;
   ...
} identifier;
```

Extensión de XDR para la Declaración de Procedimientos

ONC-RPC es un protocolo de mensajes especificado en XDR

El lenguaje especificado por RPC es idéntico al lenguaje de XDR, excepto que agrega la definición de “programa”

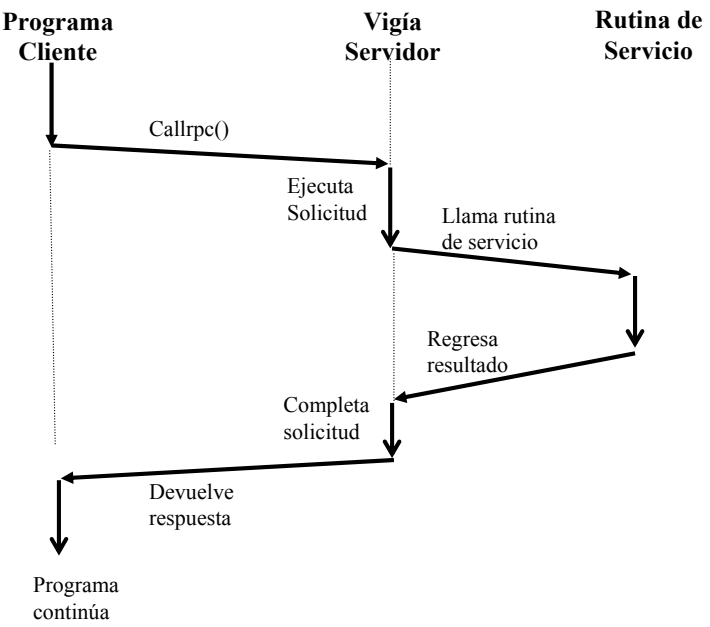
```
program-def:
     "program" identifier "{"
        version-def
        version-def *
     "}" "=" constant ";"

  version-def:
     "version" identifier "{"
         procedure-def
         procedure-def *
     "}" "=" constant ";"

  procedure-def:
     type-specifier identifier "(" type-specifier
       ("," type-specifier )* ")" "=" constant ";"
```

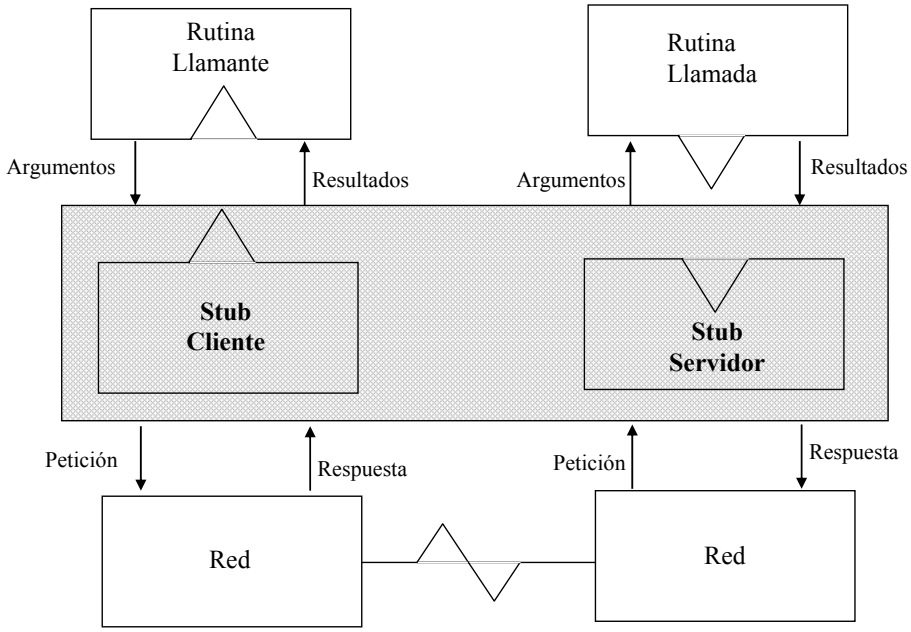
Llamados a Procedimientos Remotos

Paradigma de Comunicación entre Procesos Utilizando RPCs



Llamados a Procedimientos Remotos

Herramientas para el Desarrollo de Sistemas en Red Utilizando RPCs



Llamados a Procedimientos Remotos

Gramática del Lenguaje de Definición de Protocolos para RPCs (SUN)

<definition-list>::= <definition>; | <definition>; <definition-list>
<definition>::= <const-definition> | <enum-definition> |
<struct-definition> | <union-definition> |
<typedef-definition> | <program-definition>
<const-definition>::= const <const-ident> = <integer>
<enum-definition>::= enum <enum-ident> { <enum-value-list> }
<enum-value-list>::= <enum-value> | <enum-value>, <enum-value-list>
<enum-value>::= <enum-value-ident> | <enum-value-ident> = <value>
<struct-definition>::= struct <struct-ident> { <declaration-list> }
<declaration-list>::= <declaration>; | <declaration>; <declaration-list>
<union-definition>::= union <union-ident> switch (<declaration>)
{ <case-list> }
<case-list>::= case <value> : <declaration>; |
default : <declaration>; |
case <value> : <declaration> ; <case-list>
<typedef-definition>::= typedef <declaration>
<program-definition>::= program <program-ident>
{ <version-list> } = <value>

Llamados a Procedimientos Remotos

Gramática del Lenguaje de Definición de Protocolos para RPCs (SUN)

<version-list>::= <version> ; | <version> ; <version-list>
<version>::= version <version-ident> {
<procedure-list>
}
<procedure.list>::= <procedure> ; | <procedure> ; <procedure-list>
<procedure>::= <type-ident> <procedure-ident>
(<type-ident>) = <value>
<declaration>::= <simple-declaration> | <fixed-array-declaration> |
<variable-array-declaration> |
<pointer-declaration>
<simple-declaration>::= <type-ident> <variable-ident>
<fixed-array-declaration>::= <type-ident> <variable-ident> [<value>]
<variable-array-declaration>::= <type-ident> <variable-ident>
< <value> > | <type-ident> <variable-ident> < >
<pointer-declaration>::= <type-ident> * <variable-ident>

Llamados a Procedimientos Remotos

Gramática del Lenguaje de Definición de Protocolos para RPCs (SUN)

En virtud de que los programas escritos con esta gramática se van a traducir a C, es necesario aclarar ciertos aspectos.

• En esta gramática existen dos tipos de arreglos: de longitud fija -[ ]- y de longitud variable -< >-. En el caso de longitud variable se puede especificar una cota superior, en el caso de longitud fija siempre hay que especificar su dimensión.

• También existe el tipo de dato *string*, el cual se va a traducir como un apuntador a caracteres en C.

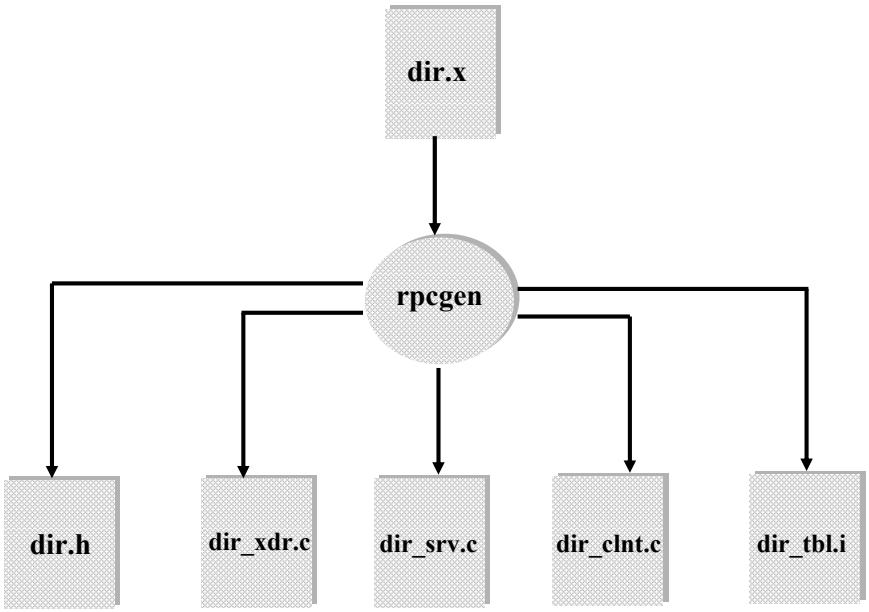
• Existe el tipo de dato opaque, este se considera como una secuencia de bytes en los cuales no se realiza ninguna transformación por los procedimientos para la representación única, aunque desde luego se debe traducir a C.

Opaque datos[512];
Opaque data<1024>;

```
char datos[520];
struct { u_int data_len;
         char *data_val
};
```

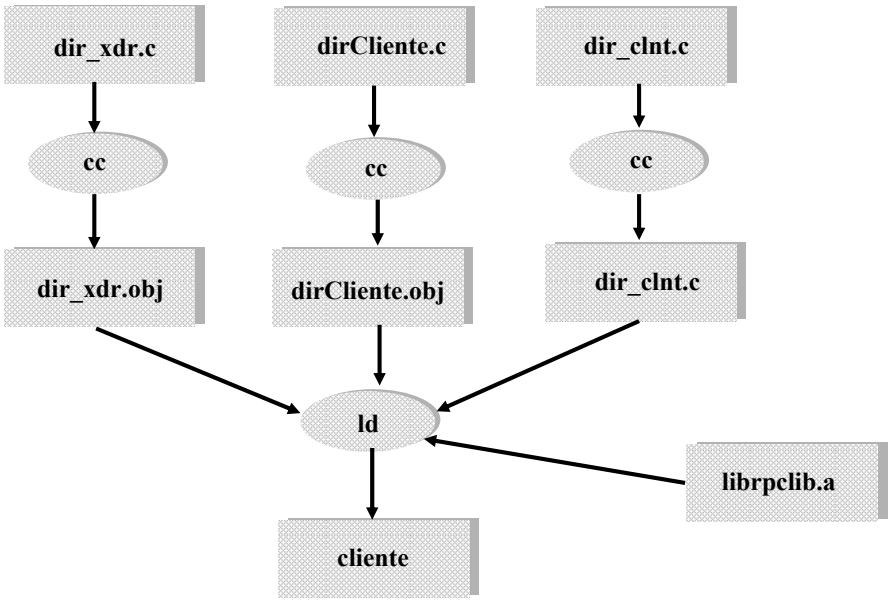
Llamados a Procedimientos Remotos

Proceso de Generación de Programas con los RPCs de SUN



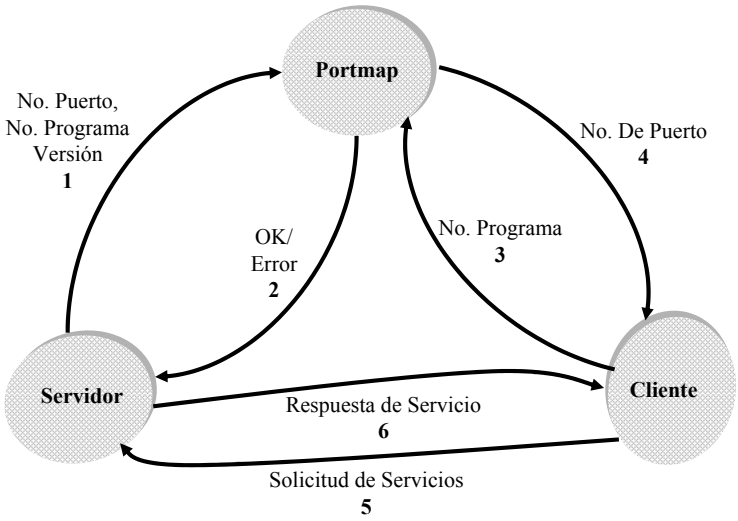
Llamados a Procedimientos Remotos

Proceso de Generación de Programas con los RPCs de SUN

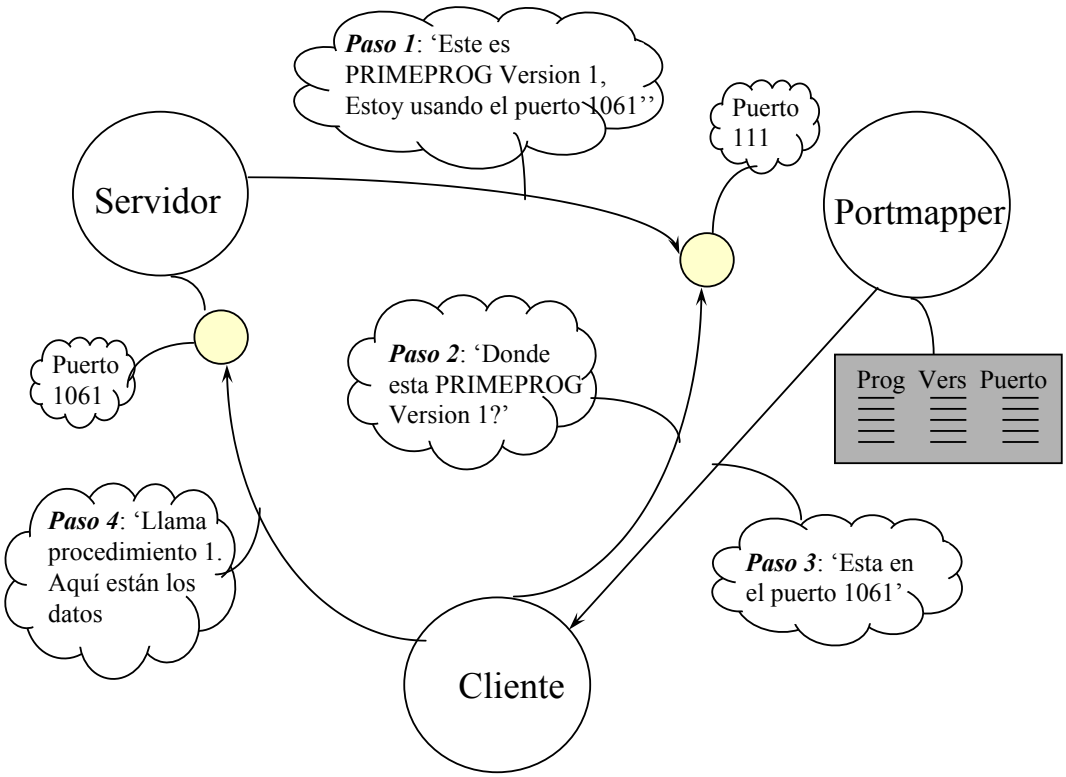


Llamados a Procedimientos Remotos

Arquitectura de los Servicios con RPCs



Registro y Localización de un Servidor RPC



Llamados a Procedimientos Remotos

Números de Programas Asignados en RPCs

Nombre	Num. Asignado	Descripción
portmap	100000	Port mapper
rstatd	100001	rstat, rup, y perfmeter
rusersd	100002	Remote users
nfs	100003	Network file systme
ypserv	100004	yp (ahora se llama NIS)
mountd	100005	mount, showmount
dbxd	100006	DBXprog(debugger)
ypbind	100007	NIS binder
walld	100008	rwall, shutdown
yppasswdd	100009	yppasswd
etherstatd	100010	Ethernet statistics

Nombre	Num. Asignado	Descripción
rquotad	100011	rquotaprog,quota,rquota
sprayd	100012	spray
selection_svc	100015	selection sevice
dbsessionmgr	100016	unify, netdbms, dbms
rexd	100017	rex, remote_exec
office_auto	100018	alice
lockd	100020	klmprog
lockd	100021	nlmprog
statd	100024	status monitor
bootparamd	100026	bootstrap
pcnfsd	150001	NFS para PC

Llamados a Procedimientos Remotos

Asignación de Números de Programas en los RPCs de SUN

Existe un número razonable de servicios creados con los RPCs de SUN Microsystems, entre los cuales destacan NFS, portmap, mount, yellowpage y otros. En vista de que el número de programas es un valor de 32 bits, SUN ha realizado una distribución de ese rango de números y es la que se muestra a continuación

Desde	Hasta	Valores asignados por
0x00000000	0x1FFFFFFF	Sun Microsystems, Inc.
0x00000000	0x3FFFFFFF	El Administrador del Sitio
0x00000000	0x5FFFFFFF	Transitorios
0x00000000	0xFFFFFFFF	Reservados

Llamados a Procedimientos Remotos

Comparación SUN/ONC y DCE/NCS

	NCS (Apollo* - DCE-OSF)	**ONC** (Sun)
Independencia de representación de datos	"El receptor corrige" Hasta 16 formatos definidos.	Representación estandarizada "XDR"
Compiladores	NIDL	RPCGEN
Autenticación		Tipo Unix/DES
Localización	Location Broker	NIS
Base instalada+	1.2 millones	400,000

* Apollo es la compañía que los propuso, pero ya desapareció. Ahora son soportados por DCE-OSF (Distrib. Comp. Env)

+ Estimaciones. No se tienen datos precisos.

Ejemplos de programas distribuidos: Notas

La aplicación RPC está constituida por:

dir.x: archivo que contiene la especificación de los protocolos.

dirServer.c: archivo que contiene las funciones que constituyen el servidor: busq_nombre_2(), dame_cadena_2(), y busq_varios_2().

dirCliente.c en este archivo se encuentra el código del cliente con las tres opciones para usar los diferentes servicios del servidor y el uso de la función clnt_create().

Así mismo se muestran los archivos: dir.h (traducción de los tipos de datos y prototipos de los servicios a partir de dir.x), mkDir (comandos a make para construir la aplicación) y directorio.dat (archivo plano con diversos registros de datos).

Especificación del Protocolo en RPCs (Stream): dir.x

```
/* Directivas al Preprocesador: Pasan Directamente  */
%#define DATABASE "directorio.dat"
/* Definicion de constantes  */
const MAX_STR=255;
/* Definicion de Estructuras  */
struct direct {
   string nombre <MAX_STR>;  /* Longitud variable */
   string apPaterno <MAX_STR>; /* char *apPaterno  */
   string apMaterno <MAX_STR>;
   int extension;
   string mensaje <MAX_STR>;
};
struct nReg{
   struct direct regs<>;
   bool hayMas;
};
typedef opaque arrDatos<2048>;

/* Definicion del Programa */

program DIRPROG{
  version DIRVERS{
     direct BUSQ_NOMBRE(string)=1; /* No. de la funcion  */
     nReg BUSQ_VARIOS(int)=2;
     arrDatos DAME_CADENA(int)=3;
  }=2;  /* Numero de version de la interfase. Puede haber varias en un mismo prog. */
}=0x2000001;  /* Numero de Programa  */
```

Archivo Generado por RPCGEN

```
#include <rpc/types.h>
#define DATABASE  "directorio.dat"
#define MAX_STR 255
struct direct { char *nombre;
                char *apPaterno;
                char *apMaterno;
                 int  extension;
                char *mensaje;
};
typedef struct direct direct;
bool_t xdr_direct();
struct nReg { bool_t hayMas;
              struct {
                     u_int regs_len;
                     struct direct *regs_val;
               } regs;
}
typedef struct nReg nReg;
bool_t xdr_nReg();
typedef struct {u_int arrDatos_len;
                char *arrDatos_val;
} arrDatos;
bool_t xdr_arrDatos();
#define DIRPROG ((u_long)0x200001)
#define DIRVERS ((u_long)2)
#define BUSQ_NOMBRE ((u_long)1)
extern direct *busq_nombre_2();
#define BUSQ_VARIOS((u_long)2)
extern nReg *busq_varios_2();
#define DAME_CADENA ((u_long)3)
extern arrDatos *dame_cadena_2();
```

Cliente en Base a RPCs (Stream): dirCliente.c

```
// Envia comandos al servidor para trabajar con archivos
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <sys/errno.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <rpc/rpc.h>  // RPCs
#include "dir.h" // estructura de los mensajes

direct *pR;
nReg *pNreg;
arrDatos *misDatos;
int i, cuantos=3;

main(int argc,char *argv[])
{
  CLIENT *cl;    /* un handle del cliente */
  char *valor;
  int  key=0;
  if(argc>=3) key=atol(argv[2]);
  if((key==1 && argc!=4) || ((key==2 || key==3) && argc!=3)
      || key <=0 || (!isdigit(argv[2][0]))) {
      printf("USO: %s nombreServidor Servicio [valorBuscado]\n",argv[0]);
      printf("\t\tEJEMPLO: dirCliente neumann 1 Maria\n\t\tSERVICIOS\n");
      printf("\t\t1. Acceder por Nombre\n");
      printf("\t\t2. Acceder varios Registros\n");
      printf("\t\t3. Acceder una Cadena de Longitud Variable\n");
      exit(1);
   }
   /* argv[1]: maquina donde esta el servicio.  DIRPROG: Numero del Programa */
   /*  DIRVERS: Version.     tcp: Protocolo     */
   if (!(cl=clnt_create(argv[1], DIRPROG, DIRVERS, "tcp"))){
          clnt_pcreateerror(argv[1]);  exit(1);
    }
```


Cliente en Base a RPCs (Stream): dirCliente.c

```
 switch (key = atol(argv[2])) {
         case BUSQ_NOMBRE:
           pR = busq_nombre_2(&argv[3],cl);
           printf("CLIENTE\nNombre\tapPaterno\tapMaterno\tExtensión\tMensaje\n");
           printf("%s\t %s\t\t %s\t\t %d\t\t %s\n", pR->nombre,pR->apPaterno,pR->apMaterno,pR-
>extension,pR->mensaje);
           break;

         case DAME_CADENA:
           cuantos=2;
           misDatos= dame_cadena_2(&cuantos,cl);
           printf("Caracteres Recibidos= %d\n",misDatos->arrDatos_len);
           for (i=0; i < misDatos->arrDatos_len; i++)
                  printf("%c", misDatos->arrDatos_val[i]);
             printf("\n");
             break;

         case BUSQ_VARIOS:
           cuantos=3;
           pNreg= busq_varios_2(&cuantos,cl);
           if(pNreg==NULL) break;

           printf("CLIENTE\nNombre\tapPaterno\tapMaterno\tExtensión\tMensaje\n");
           for(i=0, pR=pNreg->regs.regs_val; i<pNreg->regs.regs_len;
                  i++, pR= pNreg->regs.regs_val+i)
              printf("%s\t %s\t\t %s\t\t %d\t\t %s\n", pR->nombre,pR->apPaterno,pR-
>apMaterno,pR->extension,pR->mensaje);
             printf(" %s MAS DATOS\n", pNreg->hayMas?"HAY":"NO HAY");
             break;

           default:
                  printf("CLIENTE: %s: Servicio Desconocido\n",argv[0]);
   }
}
```

Servidor en Base a RPCs (Stream): dirServer.c

```
// Envia comandos al servidor para trabajar con archivos

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <sys/errno.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <rpc/rpc.h>  // RPCs
#include "dir.h" // estructura de los mensajes
FILE *fp=NULL;
char nombre[MAX_STR], apPaterno[MAX_STR], apMaterno[MAX_STR], mensaje[MAX_STR];

static direct pR={nombre,apPaterno,apMaterno,0,mensaje};
static nReg nRegs;
direct noEsta={"noEsta","noEsta","noEsta",0,"sinComentarios"};
direct arrDirect[3]= {{"Victor","Sosa","Sosa",143,"Hola1"},
{"Juan","Perez","Serna",147,"Hola2"}, {"Javier","Ortiz","hdz",156,"Hola3"}
                        };
char cadena[]= "Aracely Arroyo R 3604 Hola1 Pedro Perez G 4221 Hola2";

int leeDirectorio()
{
   char buf[MAX_STR];
   if(!fgets(buf,MAX_STR - 1,fp))
         return(0);
   sscanf(buf,"%s%s%s%d%s",
pR.nombre,pR.apPaterno,pR.apMaterno,&pR.extension,pR.mensaje);
   return 1;
}
```

Servidor en Base a RPCs (Stream): dirServer.c

```
direct *busq_nombre_2_svc(char **nombreBuscado, struct svc_req *rqstp)
{
  printf("SERVIDOR: Nombre Buscado: %s\n", *nombreBuscado);
  if(!(fp=fopen(DATABASE, "r"))) {
      printf("Error al abrir el archivo: %s\n",DATABASE);
      return((direct *)NULL);
  }
  while (leeDirectorio())
     if(!strcmp(pR.nombre, *nombreBuscado)) break;
     if (feof(fp)) {
        printf("SERVIDOR: No Encontre el Registro\n");
        fclose(fp); return(&noEsta);
  }
  fclose(fp); return(&pR);
}

arrDatos *dame_cadena_2_svc(int *nb, struct svc_req *rqstp)
{
 static arrDatos misDatos;
 printf("SERVIDOR: Me Piden %d Registros\n", *nb);
 misDatos.arrDatos_len= strlen(cadena);
 misDatos.arrDatos_val= cadena;
 return(&misDatos);
}

nReg *busq_varios_2_svc(int *nbReg, struct svc_req *rqstp)
{
   printf("SERVIDOR: Me Piden %d Registros\n",*nbReg);
   nRegs.regs.regs_len=3;
   nRegs.regs.regs_val=arrDirect;
   nRegs.hayMas=TRUE;
   return (&nRegs);
}
```

Archivo mkDir (make -f mkDir)

```
# make -f mkDir

ALL:        dir.h server cliente

server:     dir_xdr.o dir_srv.o dirServer.o dir.h
            cc -g -o server dir_xdr.o dir_svc.o dirServer.o

cliente:    dir_xdr.o dir_clnt.o dirCliente.o dir.h
            cc -g -o cliente dir_xdr.o dir_clnt.o dirCliente.o

dir.h:      dir.x
            rpcgen dir.x

dir_xdr.o:  dir_xdr.c dir.h
            cc -g -c dir_xdr.c

dir_svc.o:  dir_svc.c dir.h
            cc -g -c dir_svc.c

dir_clnt.o: dir_clnt.c dir.h
            cc -g -c dir_clnt.c

dirServer.o:dirServer.c dir.h
            cc -g -c dirServer.c

dirCliente.o:dirCliente.c dir.h
            cc -g -c dirCliente.c
```

Archivo del Directorio Telefónico (directorio.dat)

Victor	Sosa	Sosa	143	Director
Javier	Ortiz	Hernandez	160	Jefe_IS
Sonia	Sanchez	Perez	139	Secretaria
Aracely	Arroyo	Martinez	140	Asistente
Pedro	Perez	Juarez	150	Quimico
Juan	Gonzalez	Serna	147	Moviles
Rene	Santaolaya	Salgado	140	Componentes
Moises	Gonzalez	Garcia	144	Colaborativos
Jose	Ruiz	Ascencio	141	Mecatronica
Olivia	Fragoso	Diaz	148	IS

Conclusiones

- Se han presentado las características técnicas más importantes del mecanismo de Llamadas a Procedimientos Remotos (RPC, Remote Procedure Calls), específicamente de la implementación ONC

- RPC es una tecnología, tradicionalmente empleada en ambiente UNIX, que permite el desarrollo de aplicaciones cliente/servidor basadas en el paradigma procedimental.

- Con el advenimiento de implementaciones para plataforma Windows, así como para Java, se concibe a RPC como una tecnología de integración entre plataformas heterogéneas de hardware y software