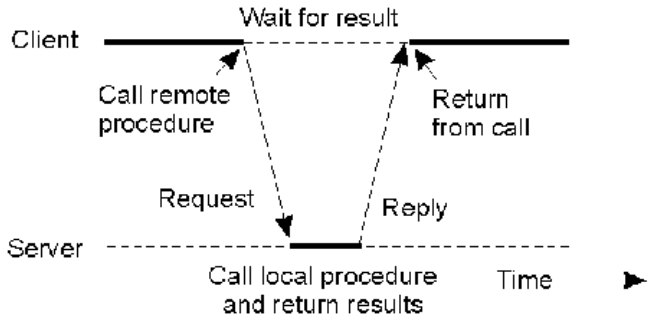
Laboratório RPC

Sistemas distribuídos em geral são baseados na troca de mensagens entre processos. Dentre os mecanismos de troca disponíveis, as Chamadas de Procedimento Remoto ou RPC (Remote Procedure Call´s) são consideradas até o momento como um pilar básico para a implementação de boa parte dos requisitos de Sistemas Distribuídos (escalabilidade, transparência etc.). Por esse motivo, faz-se necessário um estudo mais aprofundado sobre o método de programação usando RPC.

De um modo geral, pode-se dizer que as chamadas de procedimento remoto são idênticas às chamadas de procedimento local, com a exceção de que as funções chamadas ficam residentes em hosts distintos. Nesse contexto, um host executando o programa principal ou cliente aciona uma chamada de procedimento remoto (idêntico ao método de programação estruturada convencional) e ficaria aguardando um resultado. Por outro lado, outro host, denominado servidor teria as referidas funções em execução no seu espaço de memória e ficaria aguardando requisições para as mesmas. Ao chegar uma requisição, o servidor executa a função identificada e retorna os resultados para o cliente, conforme demonstra a Figura a seguir:

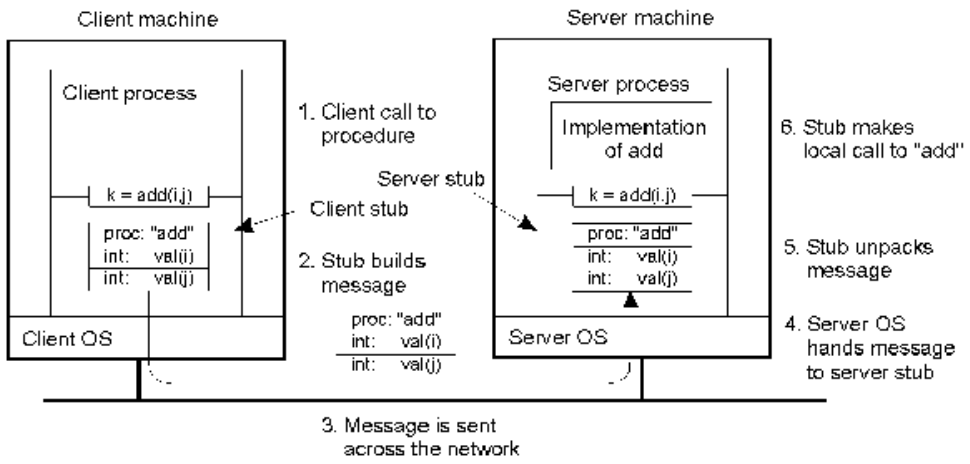


Diante do que foi dito, pode o leitor fazer alguns questionamentos tais como:

- Como o cliente consegue passar para outro host seus parâmetros de chamada?
- Como o servidor consegue individualizar cada função e saber qual delas está sendo acionada?
- Que mecanismos os lados cliente e servidor devem possuir para viabilizar chamadas remotas?

1. Princípio da comunicação RPC entre um programa cliente e um Servidor

O objetivo da biblioteca RPC é permitir ao programador uma forma de escrever o seu código de forma similar ao método adotado para a programação convencional. Para isso, a estrutura RPC define um esquema de encapsulamento de todas as funções associadas à conexão remota num pedaço de código chamado de STUB. Dessa maneira, o código do usuário terá um comportamento similar ao exemplo que está apresentado na Figura, a seguir:



Perceba que, da forma como está apresentado, existirá um STUB associado ao código do cliente e outro associado ao código do servidor. Dessa forma, o diálogo dos módulos cliente e servidor acontecerá com ajuda dos STUB´s, de acordo com a seguinte seqüência:

1. O cliente chama uma função que está implementada no stub do cliente
2. O stub do cliente constrói uma mensagem e chama o Sistema Operacional local
3. O sistema operacional do cliente envia a mensagem pela rede para o sistema operacional do host remoto
4. O sistema operacional remoto entrega a mensagem recebida para o stub instalado no servidor
5. O stub servidor desempacota os parâmetros e chama a aplicação servidora, mais especificamente, a função associada à função que estava no stub do cliente
6. O servidor faz o trabalho que deve fazer e retorna o resultado para o stub do servidor
7. O stub do servidor empacota os dados numa mensagem e chama o sistema operacional local
8. O sistema operacional do servidor envia a mensagem para o sistema operacional do cliente
9. O sistema operacional do cliente entrega a mensagem recebida para o stub do cliente
10. O stub desempacota os resultados e retorna-os para o cliente

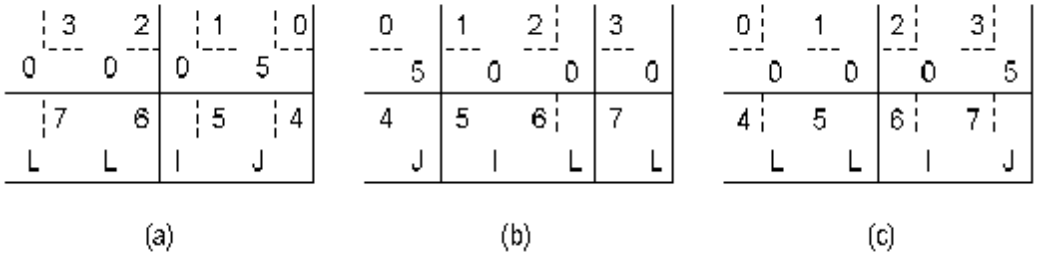
Apesar de todos esses passos e mecanismos de encapsulamento, principalmente considerando as funções de rede, a programação RPC é viável pelos seguintes motivos:

- A complexidade de encapsulamento e geração de stubs, arquivos de include e todas as chamadas de rede não são responsabilidade do programador. Em outras palavras, a biblioteca RPC contém ferramentas que auxiliam na geração dos módulos mencionados
- Caberá ao programador alterar apenas os arquivos relacionados ao “cliente” e ao “servidor”, inserindo nesses arquivos a lógica desejada. Essas alterações não incluem nenhum aspecto referente aos serviços de rede ou de como localizar o servidor.

Portanto, considerando o encapsulamento das funções de rede nos stubs cliente e servidor, a programação RPC se aproxima da programação convencional. Essa noção vai ficar mais clara à medida que os itens seguintes forem sendo compreendidos.

2. Conversão de tipos de dados entre os módulos de programa Cliente e Servidor

Uma das preocupações das implementações da biblioteca RPC é a necessidade de prover comunicação entre sistemas abertos. Ou seja, a capacidade de fazer com que funções possam ser escritas entre hosts que possuem diferentes representações internas para números (inteiro, real, etc.) e caracteres seja viabilizada. Para exemplificar, a Figura abaixo apresenta uma situação de diálogo onde um cliente instalado num microcomputador Intel quer passar a seqüência LLIJ para um servidor instalado numa SPARC Station. Conforme pode ser visto, a representação da sequência LLIJ numa máquina Intel (a) está invertida em relação ao que está representado internamente na SPARC Station após ter sido recebido (b). Nesse caso será necessário algum mecanismo de inversão desses caracteres para que o receptor compreenda o que emissor quis comunicar (a letra (c) apresenta esse modificação). No caso da figura, os números menores em cada quadro indicam o endereço de cada byte.



(a) (b) (c)

Para resolver essa questão de representação de dados entre hosts distintos existe uma biblioteca associada à biblioteca RPC denominada XDR (eXternal Data Representation), cuja funcionalidade é apresentar uma padronização entre tipos de dados de máquinas heterogêneas. Para isso, a biblioteca define uma série de tipos de dados padronizados, alguns dos quais estão apresentados na Tabela a seguir:

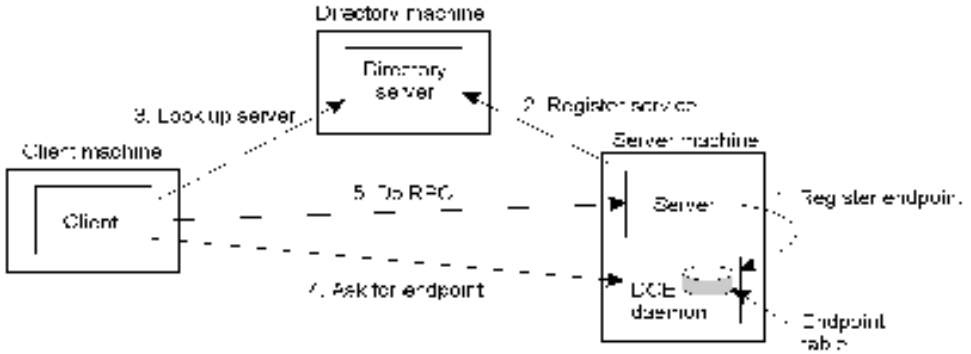
Tipo de Dado	**Tam.**	**Descrição**
Int	32 bits	Inteiro binário sinalizado de 32 bits
Unsigned int	32 bits	Inteiro binário não sinalizado de 32 bits
Bool	32 bits	Valor booleano (false ou true), representado por 0 ou 1
Enum	Arbitrário	Tipo de enumeração com valores definidos por inteiros (ex.:red=1, blue=2)
Hyper	64 bits	Inteiro sinalizado de 64 bits
...	...	....

Além desses tipos de dados, a biblioteca XDR precisa ter um conjunto de funções que permitem realizar as conversões entre os tipos de dados locais para o formato XDR. Dentre as funções existentes, algumas das mais usuais estão apresentadas na Tabela a seguir:

Função	Argumentos	Tipo de dado convertido
Xdr_bool	Xdrs, ptrbool	Booleano (int em C)
Xdr_bytes	Xdrs, ptrstr, strsize, maxsize	Byte String contado
Xdr_char	Xdrs, ptrchar	Caracter
Xdr_double	Xdrs, ptrdouble	Ponto flutuante de precisão dupla
Xdr_enum	Xdrs, ptrint	Variável de tipo de dado enumerado (um *int* em C)
Xdr_float	Xdrs, ptrfloat	Ponto flutuante de precisão simples
Xdr_int	Xdrs, ip	Inteiro de 32 bits
Xdr_long	Xdrs, ptrlong	Inteiro de 64 bits
Xdr_opaque	Xdrsptrchar, count	Bytes enviados sem uma conversão
Xdr_pointer	Xdrs, ptrobj, objsize, xdrobj	Um ponteiro (usado em listas encadeadas ou árvores)
Xdr_short	Xdrs	Inteiro de 16 bits
Xdr_string	Xdrs, ptrstr, maxsize	Uma string em linguagem C
Xdr_u_char	Xdrs, ptruchar	Inteiro não sinalizado de 8 bits
Xdr_u_int	Xdrs, ptrint	Inteiro não sinalizado de 32 bits
Xdr_u_long	Xdrs, ptrulong	Inteiro não sinalizado de 64 bits
Xdr_u_short	Xdrs, ptrushort	Inteiro não sinalizado de 16 bits
Xdr_union	Xdrsptrdiscrim, ptrunion, choicefcn, default	União discriminada

3. Como o Cliente Localiza o Servidor RPC

Toda aplicação RPC cliente servidora em geral é baseada num serviço de diretórios. Em outras palavras, o cliente, para localizar o servidor RPC remoto precisa questionar algum processo anterior que o informe sobre a porta onde determinado serviço está escutando. Na Figura a seguir está um exemplo dos passos relacionados ao processo de localização do servidor:



O serviço de diretórios então é um pré-requisito para viabilizar aplicações RPC. Em ambiente Linux, por exemplo, a implementação desse serviço de diretórios relacionados aos servidores RPC denomina-se portmapper. Dessa forma, pode-se concluir que no Linux, para executar qualquer servidor RPC será preciso antes executar o portmapper.

Falando mais especificamente sobre o RPC portmapper, este é um servidor que converte números de programa RPC em números de portas disponíveis no protocolo TCP/IP. De acordo com a Figura acima, quando um servidor RPC é executado uma das primeiras providências que ele faz é “dizer” ao serviço de diretórios (nesse caso, o portmapper) qual número de porta ele está escutando e que números de programa ele está preparado para servir. Quando um cliente deseja fazer uma

chamada RPC para um dado número de programa, ele irá primeiro contactar o portmapper na máquina servidora para determinar o número da porta onde os pacotes RPC devem ser enviados.
Um servidor RPC provê um grupo de procedures ou funções as quais o cliente pode invocar enviando uma requisição para o servidor. O servidor então invoca a procedure mencionada pelo cliente, retornando um valor quando necessário. A coleção de procedures que um servidor RPC provê é chamada de um programa e é identificado por um número de programa. No Linux, o arquivo /etc/rpc mapeia nomes de serviço para seus números de programa. Um exemplo do arquivo /etc/rpc é apresentado abaixo:

```
# RPC program number data base
# $Revision: 1.5 $        (from 88/08/01 4.0 RPCSRC)
Portmapper              100000          portmap sunrpc
Rstatd                  100001          rstat rup perfmeter
Rusersd                 100002          Rusers
Nfs                     100003          Nfsprog
Ypserv                  100004          Ypprog
Mountd                  100005          mount showmount
Walld                   100008          rwall shutdown
```

Portanto, esse arquivo só deve ser alterado se um novo serviço RPC for introduzido no servidor.

4. Usando o rpcgen para gerar os stubs do Cliente e do Servidor

Conforme já foi mencionado, junto com a biblioteca RPC existe uma ferramenta denominada rpcgen cujo objetivo é gerar, a partir de um arquivo de definição de interface (IDF – Interface Definition File), todos os módulos de software que devem estar nos lados cliente e servidor, incluindo os stubs.
De um modo geral os passos para geração de uma aplicação cliente/servidor RPC os seguintes passos devem ser considerados:

a) Identificar quais funcionalidades devem estar no programa principal e quais sub-rotinas serão acionadas no servidor. Essa percepção deve ser sistematizada no arquivo de definição de interface (Interface Definition File) apresentando na figura acima. Em outras palavras, o programador deve construir seu código usando linguagem C convencional e, a partir desse código, identificar que funções devem ser ativadas e que parâmetros devem ser passados para elas. Esses informações devem ser incluídas no arquivo de interface IDF e, a partir dele, gera-se todos os códigos da figura acima.
b) Aplicar o utilitário de geração dos módulos cliente e servidor no arquivo IDF gerado. No caso do Linux, a ferramenta de geração dos módulos da figura é o rpcgen da SUN Microsystens, considerada um padrão de facto no mercado. Esse utilitário pressupõe que o arquivo IDF tem um nome com a sufixo .x, e com base nele, os códigos são gerados em linguagem C e estão estruturados de forma que o programador possa alterá-los com o menor esforço possível.
c) Modificar os arquivos do cliente e do servidor para que atendam o objetivo desejado para a aplicação. Em princípio o programador necessita mexer apenas nesse dois arquivos, inserindo neles as lógicas presentes nas funções principal e secundárias do código que foi construído no modo convencional.
d) Compilar os códigos alterados e colocá-los em hosts cliente e servidor. No caso do rpcgen da SUN instalado em máquinas Linux, o rpcgen gera, além dos arquivos mencionados, um arquivo com diretivas de compilação para ajudar no processo de geração dos binários cliente e servidor. Esse arquivo é um Makefile.progr que deverá ser utilizado pelo utilitário make.

Com relação às modificações que um programa RPC pode ter, alguns cuidados podem ser tomados:

- Se houver alteração na lógica dos módulos cliente e/ou servidor que não comprometa a passagem de parâmetros, o correto é apenas modificar os módulos e executar novamente o comando make
- Se as alterações desejadas influenciarem no tipo de dados dos parâmetros ou na definição das funções que estão no servidor, então será necessário regerar os códigos cliente e servidor com novo uso da ferramenta rpcgen.
- Sobre o arquivo IDF, uma boa estratégia na passagem de parâmetros é encapsulá-los em uma estrutura de dados cujos campos é cada um dos parâmetros das funções. Dessa forma o cliente passa para o servidor um único parâmetro que é uma estrutura de dados. Por outro lado, o receptor deverá acessar nessa estrutura de dados recebida, o campo que lhe interessa para realizar suas tarefas.
- O processo portmapper deve estar sempre ativo no equipamento servidor
- O lado servidor deve sempre ser acionado primeiro e o cliente sempre deve ter como parâmetro de entrada o nome ou ip de localização do servidor.

Exercícios Práticos

1.Escreva um procedimento remoto que calcule remotamente imprima o desvio padrão de x dado por:

$$d = \sqrt{\frac{(x_1 - m)^2 + (x_2 - m)^2 + \cdots + (x_n - m)^2}{n - 1}}$$

onde **m** é a média do valores de **x**. Imprima o resultado usando arredondamento de 3 casas decimais.

ENTRADA DE EXEMPLO:
[10,20,30,40,50,60,70,80,90,100]

SAÍDA DE EXEMPLO:
30.277

2. Escreva um procedimento remoto que realiza o comando Mycopy, permitindo a cópia de um ou mais diretórios remotos. Implemente o cliente de modo que o usuário entre com o caminho do diretório a ser copiado. Utilize threads.

Regras Importantes:

- Os exercícios devem ser desenvolvidos nas linguagens Java, C ou C++ e deverão rodar sobre os protocolos TCP ou UDP
- O processo servidor deve obrigatoriamente rodar em um computador diferente dos processos clientes
- Cada equipe deverá ser composta por no máximo 3 integrantes
- Cada equipe tem no máximo 8 minutos para apresentar sua solução:
    - +- 05 minutos para apresentar as aplicações. Incluindo descrição do código
    - +- 03 minutos para mostrar funcionando
- Data de entrega: 29/03/2018 (no laboratório 1 do ICOMP)
- Critérios de avaliação:
    - Entrega da documentação necessária: apresentação e código funcionando
    - Sinergia do grupo no conhecimento da solução proposta
    - Funcionalidades fornecidas pelas aplicações (interface e facilidade de manipulação)

- **Dicas:**
    - Teste **exaustivamente** suas aplicações no laboratório em que ela será apresentada horas antes da apresentação
    - Cheque o tempo de sua apresentação. Lembre-se que o limite máximo de tempo é de 8 minutos
    - Qualquer alteração as regras estipuladas deverá ser discutida com o professor da disciplina