Remote Procedure Call[1]

Embora o modelo utilizado pela comunicação entre processo através de *sockets*, ou seja o modelo cliente-servidor, seja uma forma conveniente de estruturar sistemas distribuídos, ele sofre de um problema básico: toda comunicação se baseia em comandos de entrada e saída, isto é, operações do tipo *send* e *receive*.

Este problema já foi resolvido com a introdução de um conceito simples, ou seja utilizar a mesma idéia da chamada de procedimentos em uma mesma máquina, para chamar procedimentos em máquinas remotas. Este novo conceito, simples, chama-se chamada remota a procedimentos, ou seja, *remote procedure call* - RPC.

Exemplo: quando um processo na máquina A chama um procedimento na máquina B, o processo em A que chamou o procedimento é suspenso e a execução do procedimento chamado que se encontra em B é realizado. Informações podem ser enviadas da máquina A para a máquina B, através da passagem de parametros, e podem também ser retornados no resultado do procedimento. A forma como os dados sao trocados entre as máquinas nao é visível ao programador.

Apesar de parecer simples, diversos problemas existem no processo de RPC. Primeiramente, devido ao procedimento que chamou e o chamado estarem em máquinas diferentes, eles executam em espaços de endereçamento diferentes, o que pode trazer complicações. Parâmetros e resultados tem que ser passados entre os procedimentos, o que pode também ser complicado, ainda mais se as máquinas nao forem idênticas.

RPC Básico

Para melhor entender como funciona o mecanismo de RPC, é importante compreender como funciona uma chamada de procedimento tradicional. Considere a chamada abaixo:

```
count = read(fd, buf, nbytes)
```

onde `fd` é um inteiro, `buf` é uma cadeia de caracteres, e `nbytes` é um outro inteiro. Se a chamada a este procedimento for feita do programa principal, a pilha do sistema estará como mostrado na figura abaixo (a) antes da chamada. Ao fazer a chamada, os parâmetros e endereço de retorno são colocados na pilha (b). Após ter executado a operação `read` coloca o valor de retorno, por exemplo, em um registrador e retorna a execução para a posição de onde o procedimento foi chamado. Os parâmetros são então removidos da pilha, conforme mostrado na figura (c).

Neste processo é importante notar diversas coisas. Por exemplo, existem diversas formas de passar os parâmetros para o procedimento que está sendo chamado, por exemplo: passagem por valor, por referência, ou por cópia-restaura. A forma como os parâmetros são passados para

[1] Este material é extraído de [TAN95]

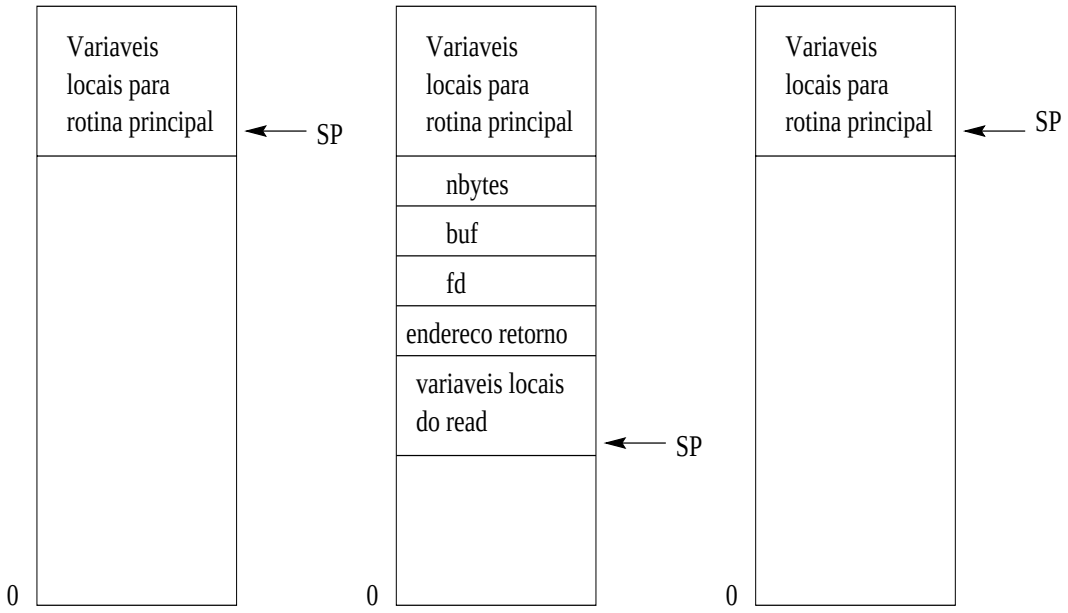

Figure 1: (a) A pilha antes da chamada de *read*. (b) A pilha quando a rotina *read* esta sendo executada. (c) A pilha após a execução da rotina *read*.

o procedimento depende da escolha do projetista de uma linguagem. Algumas vezes depende também dos tipos de parâmetros que são passados.

A idéia por trás do sistema de RPC é fazer uma chamada de procedimento remoto parecer o máximo possível com uma chamada de procedimento local. Em outras palavras se deseja que RPC seja transparente - quem chama o procedimento remoto não deve ter que se preocupar, ou saber, que o procedimento que está sendo chamado não é local. Suponha por exemplo que um programa necessite ler dados de um arquivo. O programador faz uma chamada ao procedimento *read*. No modelo tradicional, a rotina *read* é extraída de uma biblioteca pelo *linker* e inserida no programa objeto. Normalmente este procedimento é bem simples, e simplesmente coloca alguns valores nos registradores e faz uma chamada ao sistema operacional para realizar a operação de leitura. Este procedimento é na realidade um tipo de interface entre o programa e o sistema operacional.

Apesar de a rotina *read* fazer uma chamada ao sistema operacional, ela é chamada da maneira tradicional, colocando os parâmetros na pilha conforme mostrado na figura acima. Desta forma o programador não sabe que esta rotina está fazendo algo estranho.

Em RPC esta transparência é atingida de maneira similar. Os passos executados para ativar um procedimento remoto são os seguintes:

1. O procedimento cliente chama o *stub* do cliente normalmente.
2. O *stub* do cliente constrói (empacota) uma mensagem, que contém por exemplo os paramet-

ros a serem enviados para o procedimento remoto, e faz uma chamada ao sistema operacional (*kernel*, ou sistema de suporte RPC).

3. O *kernel* envia a mensagem para o *kernel* remoto.
4. O *kernel* remoto entrega a mensagem para o *stub* do servidor.
5. O *stub* do servidor desempacota a mensagem e chama o servidor.
6. O servidor executa o procedimento e retorna o resultado para o *stub*.
7. O *stub* do servidor constrói a mensagem e envia para o *kernel*.
8. O *kernel* remoto envia a mensagem para o *kernel* do cliente.
9. O *kernel* do cliente repassa a mensagem para o *stub* do cliente.
10. O *stub* do cliente desempacota o resultado e retorna para o cliente.

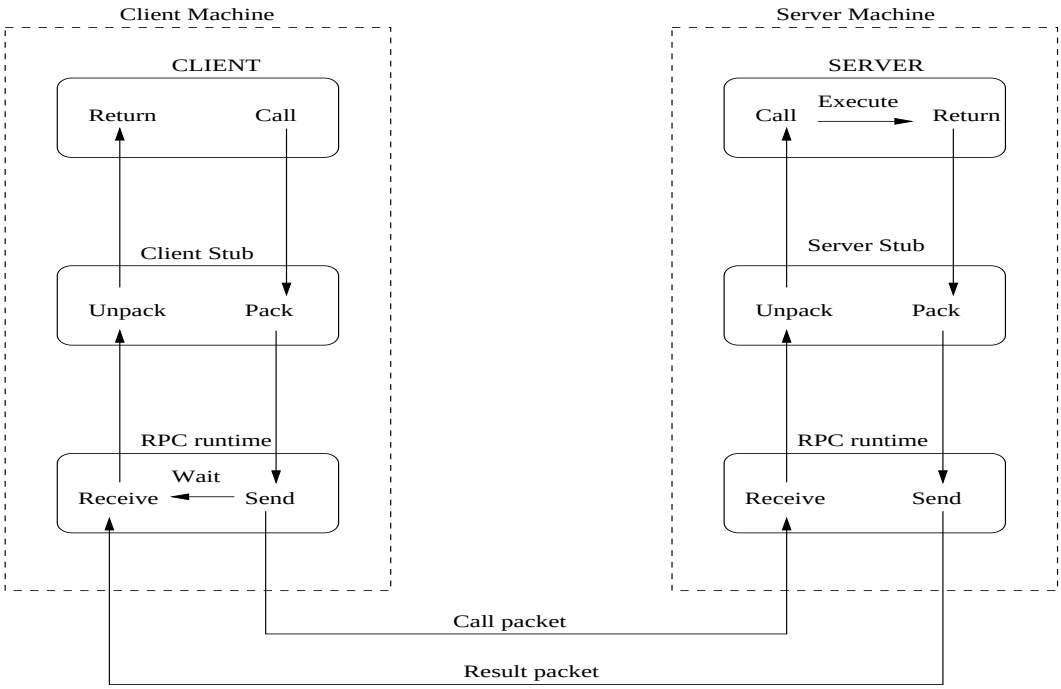

Figure 2: Sequência de passos de um mecanismo do RPC.

O processo de empacotamento dos parâmetros a serem enviados para o procedimento não é tão direto como parece. Este procedimento é normalmente chamado de *parameter marshaling*. Em sistemas remotos onde as arquiteturas das máquinas é a mesma o problema nao é tao grave. Caso as máquinas sejam diferentes, então cada uma pode ter seu próprio sistema de representação para

números, caracteres ou outros itens de dados. Por exemplo, computadores de grande porte IBM utilizam o sistema EBCDIC, enquanto IBM PCs utilizam ASCII. Outros problemas similares podem acontecer com a representação de números.

Para maiores informações a respeito de implementação e problemas quando falhas acontecem durante a chamada remota a um procedimento podem ser encontradas em [TAN95], páginas 72-98.

[TAN95] A. S. Tanenbaum. *Distributed Operating System.* Prentice Hall, New Jersey, USA, 1995.