Sistemas Distribuídos
RPC

Disciplina: Sistemas Distribuídos
Prof.: Edmar Roberto Santana de Rezende

Faculdade de Engenharia de Computação
Centro de Ciências Exatas, Ambientais e de Tecnologias
Pontifícia Universidade Católica de Campinas

Remote Procedure Call

Necessidade

- Modelo Cliente-Servidor
  - toda a comunicação é feita através de operações de I/O:
    - *send* → saída de dados
    - *receive* → entrada de dados
- É interessante tornar a computação distribuída parecida com a computação centralizada
  - transparência: facilita a programação
    - operações de I/O não são uma forma de se fazer isto
- Paper de Birrel and Nelson (1984)
  - introduziu uma forma totalmente diferente de tratar este problema

Remote Procedure Call

Birrel and Nelson (1984)

- Proposta:
  - fazer com que programas sejam capazes de chamar procedimentos localizados em outras máquinas
    - quando um processo na máquina A chama um procedimento na máquina B:
      - o processo A é suspenso
      - a execução do procedimento chamado é realizada em B
      - informações podem ser transportadas nos parâmetros
      - informações podem retornar no resultado do procedimento
  - nenhuma passagem de mensagem ou I/O é visível para o programador
- Método conhecido como: **RPC (Remote Procedure Call)**

Remote Procedure Call

Operação RPC básica

- Chamada de procedimento convencional

  count = read(fd, buf, nbytes)

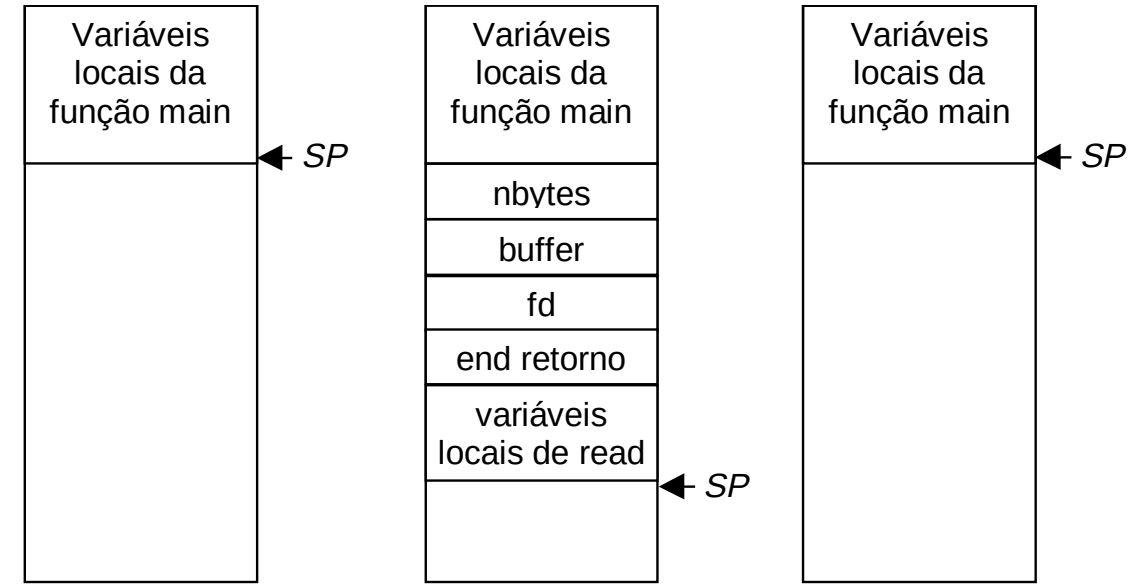
- Parâmetros:
  - primeiro: um descritor de arquivo
  - segundo: um array de caracteres
  - terceiro: um inteiro
- Parâmetros são empilhados de modo a que o primeiro argumento esteja no topo da pilha
  - facilita o acesso aos dados

Remote Procedure Call

Operação RPC básica

- Chamada de procedimento convencional

count = read(fd, buf, nbytes)



(a) antes chamada da função read

(b) durante execução da função read

(c) após chamada da função read

Remote Procedure Call

Passagem de parâmetros

- A forma como os parâmetros são tratados define 3 tipos de passagem de parâmetros:
  - Chamada por valor (*call-by-value*)
  - Chamada por referência (*call-by-reference*)
  - Chamada por cópia e restauração (*call-by-copy/restore*)
- Chamada por valor (*call-by-value*):
  - o valor do parâmetro é copiado para a pilha
  - o procedimento pode alterar o valor da forma que desejar que tal mudança não será refletida na variável original após o retorno da função

Remote Procedure Call

Passagem de parâmetros

- Chamada por referência (*call-by-reference*):
  - o endereço do parâmetro é copiado para a pilha
  - quando o procedimento alterar uma variável, esse valor será mantido no retorno da chamada uma vez que a alteração é feita diretamente na variável original
- Chamada por cópia e restauração (call-by-copy/restore):
  - a variável é copiada para a pilha (como se fosse em uma chamada por valor) e no retorno da chamada o valor alterado sobrescreve o valor original (como se fosse uma chamada por referência)

Remote Procedure Call

Passagem de parâmetros

- A chamada por cópia e restauração normalmente atinge o mesmo efeito da chamada por referência
  - Em algumas situações a semântica não é a mesma
    - Exemplo: quando uma mesma variável estiver presente múltiplas vezes na lista de parâmetros
- Implementações RPC
  - via de regra usa-se chamada por valor ou por cópia e restauração

Remote Procedure Call

Passagem de parâmetros

- A decisão por qual mecanismo de passagem de parâmetros utilizar normalmente é feita pelos projetistas da linguagem:
  - C:
    - inteiros e outros tipos escalares são sempre passados por valor
    - arrays são sempre passados por referência
  - Pascal:
    - programador pode escolher que mecanismo quer usar para cada parâmetro
    - o padrão é utilizar chamada por valor
    - chamadas por referência são feitas inserindo a palavra **var** antes do parâmetro
  - Alguns compiladores Ada usam passagem por cópia e resturação

Remote Procedure Call

Operação RPC básica

- A idéia por trás do RPC é tornar uma chamada remota de procedimento o mais parecida possível com uma chamada local
  - Transparência
    - o processo que realiza a chamada de procedimento não deve estar ciente do fato do procedimento ser executado em outra máquina
    - Exemplo:

      Suponha que um programa precisa ler dados de um arquivo.

      O programador utiliza uma chamada *read* no código do programa para obter os dados.
    - Em essência, o procedimento *read* é um tipo de interface entre o código do usuário e o sistema operacional

Remote Procedure Call

Operação RPC básica

- O mecanismo de RPC alcança a transparência de forma análoga
  - Quando o procedimento *read* é executado:
    - uma versão diferente de *read* é utilizada

      → ***stub cliente***

      - empacota os parâmetros e pede ao kernel que envie a mensagem ao servidor
      - realiza uma chamada *send* para enviar os dados
      - em seguida realiza uma chamada *receive*, bloqueando a si mesmo até receber a resposta

Remote Procedure Call

Operação RPC básica

- Quando a mensagem chega no servidor:
  - o kernel passa a mensagem para o ***stub servidor*** que está associado ao servidor atual

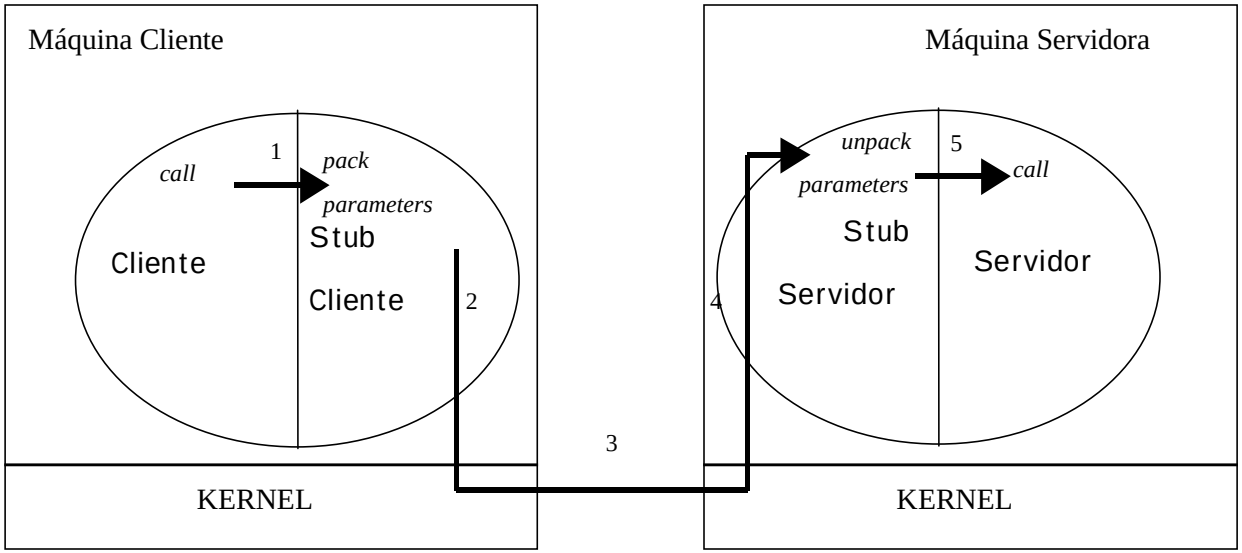
    → ***stub servidor***

    - faz uma chamada *receive* e fica bloqueado aguardando por mensagens
    - desempacota os parâmetros da mensagem
    - chama o procedimento servidor da forma tradicional
- Do ponto de vista do servidor, a chamada está sendo feita diretamente pelo cliente
  - o servidor executa o procedimento e retorna o resultado da maneira tradicional

Remote Procedure Call

Operação RPC básica

- Execução de um procedimento remoto
  - envio dos dados



Remote Procedure Call

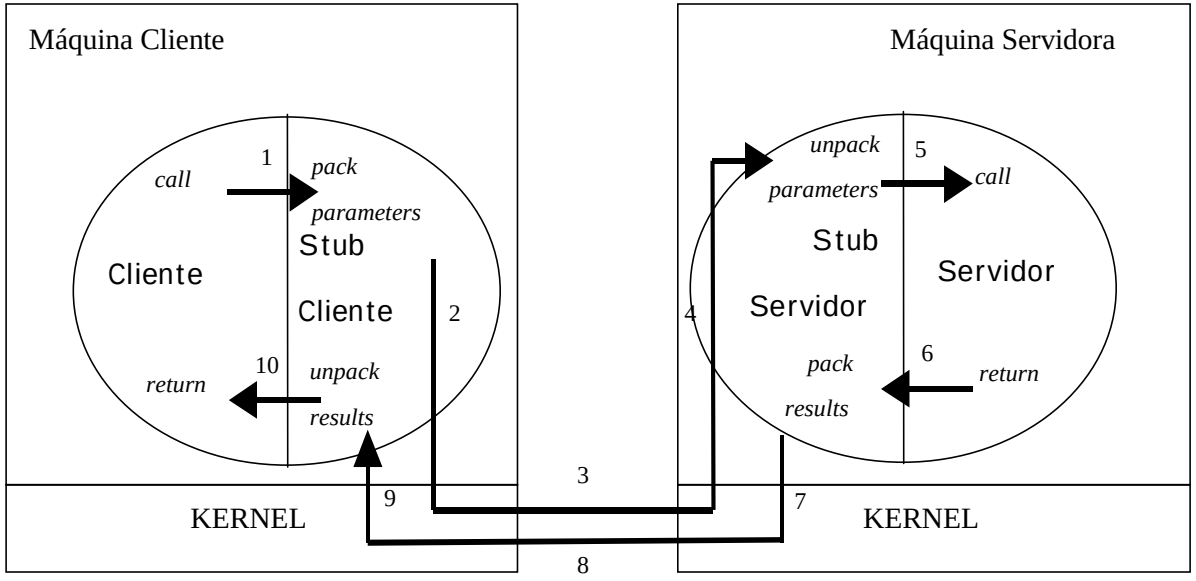
Operação RPC básica

- Quando o **stub servidor** obtém o controle novamente, após a chamada ter se completado:
  - empacota o resultado em uma mensagem
  - realiza uma chamada *send* para enviar a resposta ao cliente
- Quando a mensagem retorna a máquina cliente:
  - o kernel identifica que a mensagem está endereçada para o processo cliente
    → para a parte **stub** do processo, mas o kernel não sabe disso
  - a mensagem é copiada para o buffer de recepção e o cliente é desbloqueado
  - o **stub cliente** verifica a mensagem
    → desempacota o resultado
    → copia o resultado para quem o chamou

Remote Procedure Call

Operação RPC básica

- Execução de um procedimento remoto
  - envio e recebimento dos dados



Remote Procedure Call

Operação RPC básica

- Passos:
  1. O procedimento remoto chama o stub cliente da forma normal, isto é, como se fosse um procedimento local qualquer
  2. O stub cliente constrói a mensagem e passa ao kernel
  3. O kernel envia a mensagem ao kernel remoto
  4. O kernel remoto entrega a mensagem ao stub servidor
  5. O stub servidor desempacota os parâmetros e chama o servidor
  6. O servidor realiza o trabalho solicitado e envia o resultado ao stub servidor
  7. O stub servidor empacota o resultado em uma mensagem e passa para o kernel
  8. O kernel remoto envia mensagem ao kernel cliente
  9. O kernel do cliente entrega a mensagem ao stub cliente
  10. O stub desempacota o resultado e retorna ao cliente

Remote Procedure Call

Passagem de parâmetros

- Função do stub cliente:
  1. obter os parâmetros do cliente
  2. empacotá-los em uma mensagem
  3. enviá-los ao stub servidor

  → Este procedimento não é tão simples quanto parece

- *Parameter Marshaling*:
  - empacotamento de parâmetros em uma mensagem

Remote Procedure Call

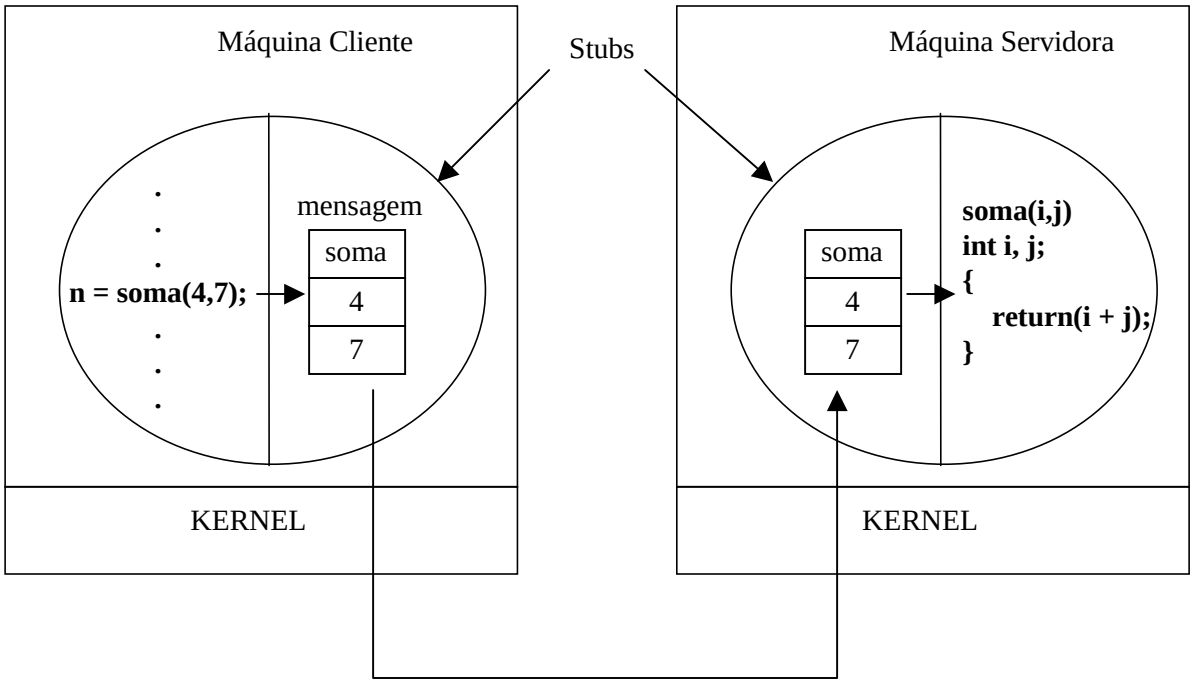
Passagem de parâmetros

- Considere um procedimento remoto:
  - soma(i, j);
    - recebe dois inteiros como parâmetro
    - retorna a soma aritimética dos parâmetros
  - Quando o procedimento é executado:
    - o stub cliente obtém os parâmetros e coloca em uma mensagem:
      - → os parâmetros
      - → o nome ou o número do procedimento chamado
  - Quando a mensagem chega no servidor:
    - o stub servidor examina a mensagem e qual procedimento será necessário

Remote Procedure Call

Passagem de parâmetros

- Executando a soma remotamente



Remote Procedure Call

Passagem de parâmetros

- O servidor pode suportar outros procedimentos:
  - subtração
  - multiplicação
  - divisão
  - pode selecionar o procedimento (if, switch)
    - → dependendo do 1º campo da mensagem

1. Servidor finaliza a execução
2. Stub servidor obtém novamente o controle
   - → obtém o resultado (retornado pelo servidor)
   - → empacota o resultado em uma mensagem
   - → envia a mensagem para o stub cliente
3. Stub cliente recebe a mensagem
   - → desempacota a mensagem
   - → retorna o valor para o procedimento cliente

Remote Procedure Call

Passagem de parâmetros

- Não há problemas se:
  - as máquinas cliente e servidora são idênticas
  - todos os parâmetros e resultados são tipos escalares
    - → inteiros, caracter, booleano
- Em um sistema distribuído:
  - é comum a existência de diversos tipos de computadores
    - → cada computador possui sua própria representação para números, caracteres, e outros tipos de dados
  - Ex:
    - Mainframes IBM: usam codificação de caracteres EBCDIC
    - PCs IBM: usam codificação de caracteres ASCII

    → Não é possível passar um caracter como parâmetro de um cliente PC IBM para um servidor Mainframe IBM com o esquema anterior

Remote Procedure Call

Passagem de parâmetros

- Problemas semelhantes podem ocorrer com:
  - representação de números inteiros
    - complemento de 1 vs. complemento de 2
  - representação de números em ponto flutuante

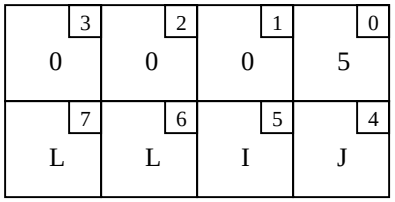
- Problemas adicionais:
  - Alguns computadores numeram seus bytes da direita para a esquerda
    - Intel 486
      → **little endian**
  - Outros numeram da esquerda para a direita
    - Sun SPARC
      → **big endian**

Remote Procedure Call

Passagem de parâmetros

- Considere um servidor com 2 parâmetros:
  - um inteiro: 5
  - uma string de 4 caracteres: “JILL”

→ cada um dos parâmetros é uma palavra de 32 bits



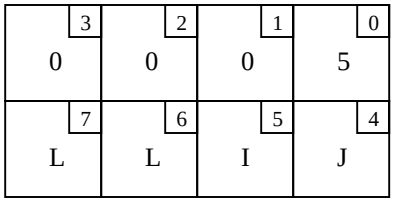
(a) Mensagem original no 486

Remote Procedure Call

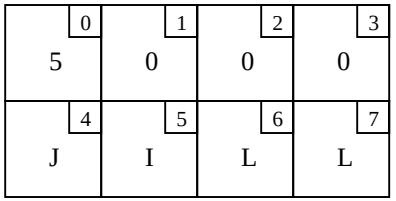
Passagem de parâmetros

- A mensagem é transferida byte por byte:
  - o primeiro byte enviado é o primeiro a chegar

- Quando o stub servidor lê os parâmetros obtém:
  - um inteiro: 83.886.080 (5×2^{24})
  - uma string: “JILL”



(a) Mensagem original no 486

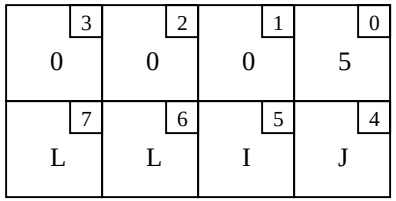


(b) Mensagem depois de recebida na SPARC

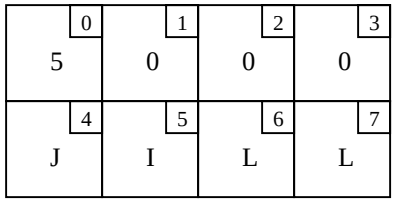
Remote Procedure Call

Passagem de parâmetros

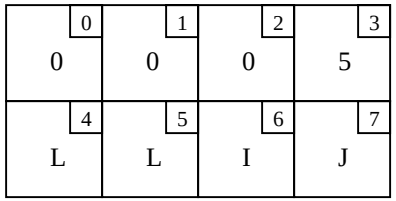
- Solução:
  - Inverter os bytes de cada palavra após o recebimento
- Após a inversão:
  - um inteiro: 5
  - uma string: “LLIJ”



(a) Mensagem original no 486



(b) Mensagem depois de recebida na SPARC



(c) Mensagem depois de ser invertida

Remote Procedure Call

Passagem de parâmetros

- Problema:
  - inteiros são representados em uma ordem de bytes diferente
    - → mas as strings não!
- Observação:
  - sem informações sobre o que é uma string e o que é um inteiro fica impossível corrigir o engano

Remote Procedure Call

Passagem de parâmetros

- Detalhe:
  - essa informação já está implicitamente disponível
    - → os itens na mensagem correspondem ao identificador do procedimento e seus parâmetros
    - → ambos, cliente e servidor, conhecem os tipos dos parâmetros
  - procedimento com n parâmetros:
    - mensagem possui n+1 campos
      - → 1 para o identificador do procedimento
      - → 1 para cada parâmetro

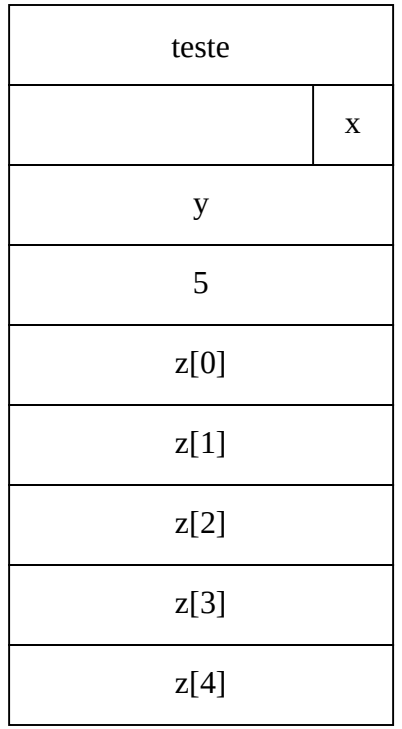
Remote Procedure Call

Passagem de parâmetros

- Exemplo:
  - procedimento: teste
  - 3 parâmetros
    - um caracter
    - um número em ponto flutuante
    - um array com 5 inteiros
- Regras:
  - **caracter:** byte mais a direita de uma palavra
  - **ponto flutuante:** uma palavra inteira
  - **array:** um grupo de palavras igual ao tamanho do array, precedido por uma palavra indicando o tamanho do array

→ Conhecidos os tipos e as regras é possível efetuar as conversões necessárias

```
teste (x, y, z)
char x;
float y;
int z[5];
{
...
}
```



Remote Procedure Call

Passagem de parâmetros

- Ainda há um problema em aberto:
  - Como as informações devem ser representadas?
- Solução:
  - Estabelecer um padrão para transmissão
    - forma canônica para inteiros, caracteres, booleanos, números em ponto flutuante, etc
    - todos os emissores devem converter sua representação interna para esta forma padrão durante o marshaling
    - Ex: inteiros (complemento de 2), caracteres (ASCII), booleanos (0 e 1) e ponto flutuante (formato IEEE), tudo armazenado como little endian
  - O stub servidor não precisa mais se preocupar com a ordem dos bytes do cliente

    → a ordem será fixa, independente do hardware

Remote Procedure Call

Passagem de parâmetros

- Problema com esta abordagem:
  - algumas vezes ela é ineficiente!
  - Cenário:
    - cliente big endian conversando com servidor big endian
  - De acordo com as regras:
    - o cliente deve converter tudo na mensagem para little endian
    - o servidor deve desfazer a conversão quando receber a mensagem

    →Requer duas conversões

    →Nehuma conversão é necessária!

Remote Procedure Call

Passagem de parâmetros

- Segunda abordagem:
  - o cliente usa seu formato nativo
  - o primeiro byte da mensagem indica o formato utilizado

  - Funcionamento:

    1.a) cliente little endian constrói mensagem little endian

    1.b) cliente big endian constrói mensagem big endian

    2) stub servidor examina o primeiro byte e decide se deve ou não converter

    - outras conversões são feitas da mesma forma:
      → complemento de 1 e complemento de 2
      → EBCDIC e ASCII

Remote Procedure Call

Geração de Stubs

- Questão:
  - Como obter os stubs?
  - Em vários sistemas baseados em RPC os stubs são gerados automaticamente:
    - tendo a especificação do procedimento do servidor e as regras de codificação
      → o formato da mensagem pode ser bem definido
    - compilador lê a especificação do servidor:
      → gera um stub cliente que empacota os parâmetros em um formato pré-determinado
      → gera um stub servidor que desempacota os parâmetros e chama o servidor
    - Reduz as chances de erros
    - Torna o sistema transparente em relação a diferentes representações de dados

Remote Procedure Call

Ponteiros

- Questão:
  - Como passar ponteiros como parâmetro?
    1) substituindo chamadas por referência por chamadas por cópia e restauração:

       Ex: **read(fd, buffer, nbytes)**

       buffer - apontador para um array de caracteres

       → o stub cliente copia o array na mensagem e envia ao servidor

       → o servidor utiliza um apontador para o array do stub servidor

       → o stub servidor envia o array para o stub cliente

       → o stub cliente sobrescreve o array no cliente

    2) indicando se o parâmetro é de saída ou de entrada

       → uma das cópias pode ser eliminada