

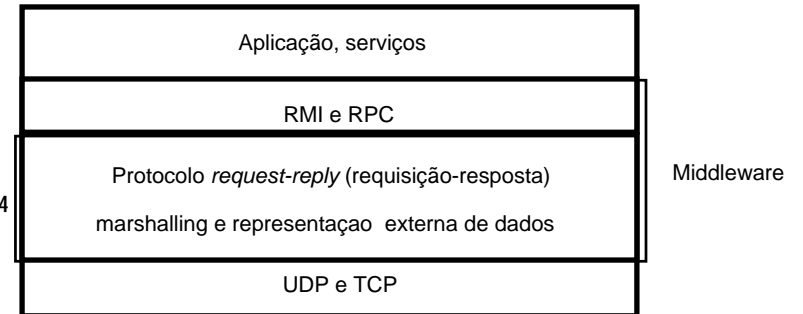
Sistemas Operacionais Distribuídos e de Redes

Comunicação Interprocessos

Aula 04

Contextualização

Aula 04



Introdução

- ❑ Sistema distribuído implica em comunicação entre processos
 - ▶ Compartilhamento de memória (variáveis ou área de memória comuns)
 - ▶ Troca de mensagens
 - ▶ Cópia de dados de um espaço de memória a outro
- ❑ Passagem de mensagem
 - ▶ Conjunto de primitivas para comunicação entre processos baseado em um protocolo de troca de mensagens
 - ▶ Esconde detalhes de protocolos de comunicação → interface mais simples
 - ▶ Tratamento de heterogeneidade
 - ▶ Exemplos: MPI, PVM, sockets, Java RMI, etc...

Características desejáveis

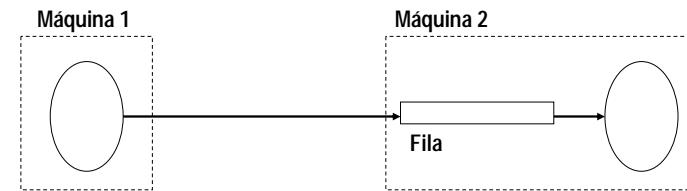
- ❑ Simplicidade e clareza de uso
- ❑ Semântica uniforme
 - ▶ Processos locais e remotos usam a mesma interface de comunicação
- ❑ Eficiência:
 - ▶ Baixo sobrecurso de processamento
 - ▶ Minimizar custo de estabelecimento/manutenção de conexões
 - ▶ Uso de *piggybacking* para eliminar mensagens adicionais
- ❑ Confiabilidade:
 - ▶ Garantir a entrega das mensagens em presença de falhas
 - ▶ Eliminar (tratar) replicação de mensagens
 - ▶ Ordenamento de mensagens

Características desejáveis (*cont.*)

- ❑ Correção (associado a comunicação em grupo)
 - ▶ Atomicidade: todos ou nenhum recebem
 - ▶ Entrega ordenada: mensagens chegam na mesma ordem a todos processos
 - ▶ Garantia de entrega
- ❑ Flexibilidade
 - ▶ Nem todas as aplicações tem as mesmas necessidades
- ❑ Segurança
 - ▶ Autenticação dos pares
 - ▶ Codificação das mensagens
- ❑ Portabilidade
 - ▶ Aplicações e da própria biblioteca (implica em heterogeneidade)

Modelo de comunicação

- ❑ Duas primitivas básicas: *send* e *receive*
 - ▶ Duas semânticas: bloqueante e não bloqueante
- ❑ Envio de mensagem (*send*)
 - ▶ Inclusão da mensagem em uma fila remota
- ❑ Recepção de mensagem (*receive*)
 - ▶ Retirada da mensagem em um fila local



Semântica de sincronização *send* e *receive*

- ❑ Semântica do *send*
 - ▶ Bloqueante: o processo do *send* espera a execução do *receive* correspondente
 - ▶ Não bloqueante: o processo fonte prossegue com a execução tão logo a mensagem seja enviada (bufferizada)
- ❑ Semântica do *receive*
 - ▶ Bloqueante: o processo destino é bloqueado até que a mensagem seja recebida
 - ▶ Não bloqueante:
 - ▶ Processo destino retorna da chamada tão logo seja alocada uma área de memória para a recepção da mensagem
 - ▶ Necessidade de saber que a mensagem chegou (notificação)
 - ▶ Polling: primitivas do tipo *test*
 - ▶ Interrupção (signal)

Comunicação síncrona *versus* assíncrona

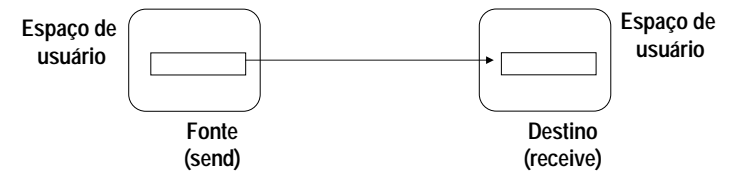
- ❑ Comunicação:
 - ▶ Síncrona: processos usam versão bloqueante das primitivas
 - ▶ Assíncrona: processos usam versão não bloqueante das primitivas
- ❑ Desvantagens da comunicação síncrona
 - ▶ Limitação da concorrência
 - ▶ Possibilidade de deadlocks
- ❑ Vantagens da comunicação síncrona
 - ▶ Programação simples
- ❑ Vantagens e desvantagens da assíncrona são *dual* da síncrona

Bufferização

- ❑ Relacionado com aspectos de sincronização
 - ▶ Transmissão: cópia da mensagem de um espaço de endereçamento a outro
- ❑ Tipos de bufferização:
 - ▶ *Null-buffer*
 - ▶ *Unbounded buffer*
 - ▶ *Single message*
 - ▶ *Finite bound buffer (multiple message)*

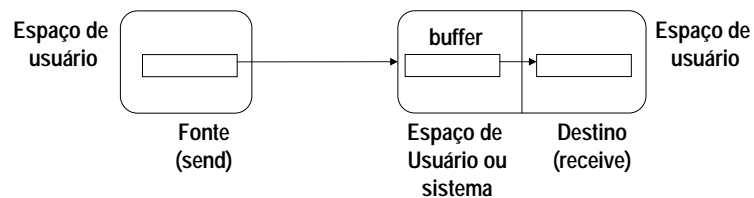
Null-buffer

- ❑ Sem espaço para armazenamento temporário
- ❑ Duas opções:
 - ▶ Mensagem é mantida no espaço de usuário do emissor até que o receptor execute o *receive* correspondente e aloque área em seu espaço
 - ▶ Mensagem (*send*) é descartada: implica em retransmissão e *ack*
- ❑ Comunicação síncrona
- ❑ Mecanismo de *rendez-vous*



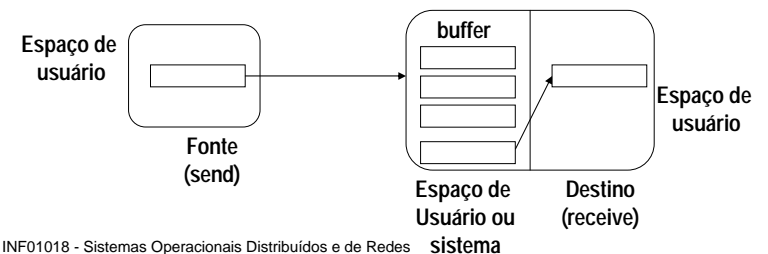
Single message buffer

- ❑ Idéia é liberar o emissor o mais rapidamente possível
 - ▶ Desbloqueado tão logo a mensagem seja copiada para o buffer de recepção e não mais dependa da execução do processo destino
- ❑ Comunicação síncrona



Unbounded buffer e finite-bound buffer

- ❑ Fonte não espera o destino estar pronto para receber mensagem
- ❑ Comunicação assíncrona
- ❑ Bufferização é infinita é impossível → alocação certa capacidade
 - ▶ Problema é quando ocorre overflow
 - ▶ Erro de comunicação
 - ▶ Controle de fluxo: bloqueia fonte (problema de troca de semântica)

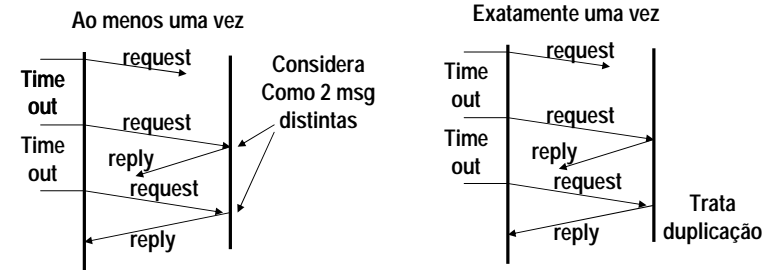


Desvantagens da bufferização

- ❑ Gerenciamento dos buffers
 - ▶ Criação, distribuição e manipulação
- ❑ Problemas de proteção de acesso
- ❑ Eventos “catastróficos”
 - ▶ O que fazer com no caso de *crash* da máquina ou do processo que mantém o buffer?

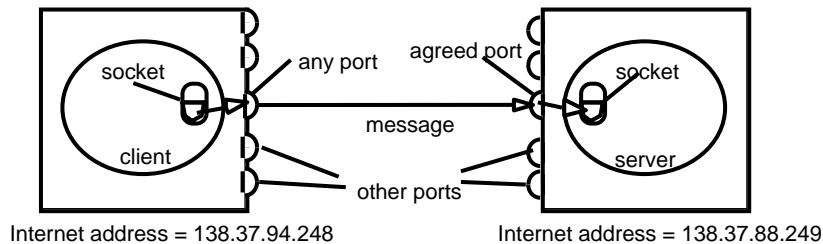
Confiável versus não confiável

- ❑ Mensagens podem ser duplicadas, entregues fora de ordem ou perdidas
 - ▶ Primitivas não confiáveis: não tratam esses problemas
 - ▶ Primitivas confiáveis: tratam esses problemas (ack, time-out, controle erro, etc)
- ❑ Duas semânticas:



Estudo de caso: Protocolos Internet e API Sockets

- ❑ TCP e UDP usam a abstração de *sockets*
- ❑ Define um ponto da comunicação {IP, Porta}
 - ▶ Uma porta não pode ser compartilhada por vários processos (exceto *multicast*)
 - ▶ Vários processos podem enviar para uma mesma porta



UDP

- ❑ Sem confirmação ou retentativas
- ❑ Primitivas *send/receive* em versões não bloqueantes e bloqueantes
 - ▶ Time-out pode ser associado as versões bloqueantes para evitar esperas por tempo indeterminado
- ❑ *Receive* não especifica a origem da mensagem (*receive any*)
- ❑ Tamanho da mensagem deve ser explicitado
- ❑ Modelo de falhas:
 - ▶ Por omissão
 - ▶ Ordenamento

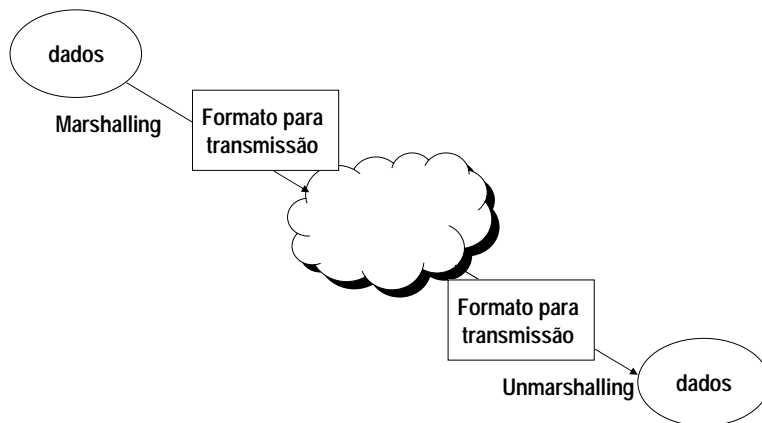
TCP

- ❑ Não há definição do tamanho da mensagem (*stream*)
- ❑ Confiabilidade:
 - ▶ Orientado a conexão
 - ▶ Garantia de entrega, ordenamento, controle de duplicação e fluxo
- ❑ Comunicação bidirecional: socket possui um par de streams (in e out)
- ❑ Modelo de falhas:
 - ▶ TCP garante integridade (perdas, detecção/rejeição de duplicados)
 - ▶ Validade: timeout+retransmissões para garantir a entrega
 - ▶ Só que conexões podem ser quebradas por timeouts

Representação de dados

- ❑ Informação (dados) é representada em uma estrutura de dados
 - ▶ Na comunicação informação é igual a uma seqüência de bytes
 - ▶ Necessidade de conversão estrutura de dados ↔ seqüência de bytes
- ❑ Problema:
 - ▶ diferentes sistemas = diferentes estruturas de dados
 - ▶ Ex: código ASC versus Unicode, big endian versus little endian, formatos ponto flutuante, etc
- ❑ Solução:
 - ▶ Conversão para um formato de dados acordado (*external data representation*)
 - ▶ Enviar no formato do emissor e incluir informação sobre o formato empregado

Princípio representação externa de dados



Marshalling

- ❑ Processo de converter uma coleção de dados e organizá-los em um formato próprio para transmissão
 - ▶ *Unmarshalling* é o processo contrário
- ❑ Ideal é que seja feito sem envolvimento explícito da aplicação
 - ▶ Responsabilidade do *middleware*
- ❑ Duas técnicas básicas:
 - ▶ Conversão dos dados para um formato binário
 - ▶ Conversão dos dados para um formato texto (ASC II)
- ❑ Três estudos de caso: Corba, serialização Java e XML

Estudo de caso: Corba *Common Data Representation*

- ❑ Define 15 tipos básicos de dados e 6 construtores para tipos mais complexos:
 - ▶ Básicos: short, long, unsigned short, float, double, boolean, char, etc
 - ▶ Construtores: sequence, string, array, struct, enumerated, union
- ❑ Dados são descritos através dos tipos e o middleware executa o (un)marshalling via sua IDL (*Interface Description Language*)
- ❑ Representação em *big endian* e *little endian*
 - ▶ Valores enviados na representação da fonte com sua indicação
- ❑ Baseado na conversão para um formato binário

Estudo de caso: serialização de objetos java

- ❑ Transformação de um objeto para uma forma “seqüência de bytes”
 - ▶ Comunicação de objetos via sockets (RMI)
 - ▶ Armazenamento em disco (*lightweight persistence*)
- ❑ Como é disponibilizado ao usuário:
 - ▶ *ObjectOutputStream* e *ObjectInputStream* + métodos *writeObject/readObject*
 - ▶ Classe do objeto implementa a interface *serializable*
 - ▶ Métodos *defaultWriteObject* e *defaultReadObject*
- ❑ Formato binário
 - ▶ Nome da classe
 - ▶ Assinatura da classe
 - ▶ Instâncias das variáveis

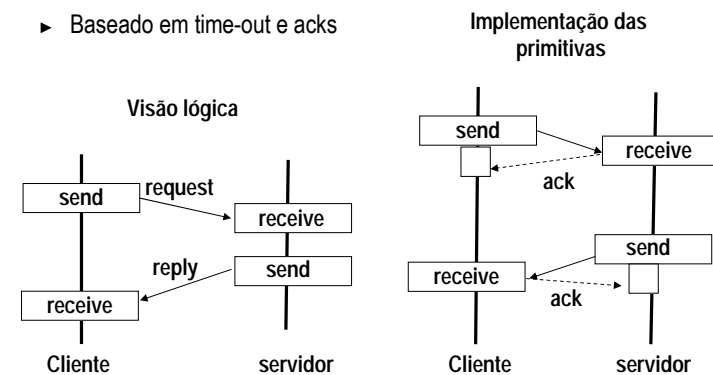
Estudo de caso: *eXtensible Markup Language* (XML)

- ❑ Codificação textual para descrever estrutura e aparência
- ❑ Baseada em marcas (tags)
 - ▶ Descrevem a estrutura lógica dos dados
 - ▶ Associação por atributo-valor

```
<person id="123456789">  
    <name>Smith</name>  
    <place>London</place>  
    <year>1934</year>  
    <!-- a comment -->  
</person >
```

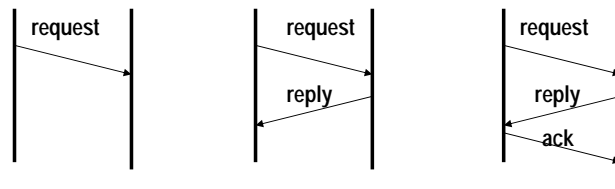
Aspectos de implementação de primitivas

- ❑ Sistema de comunicação pode ser projetado para suportar um certo nível de confiabilidade
 - ▶ Baseado em time-out e acks



Protocolos

- ❑ *Request*
 - ▶ Servidor não oferece retorna algum e cliente não precisa de confirmação
- ❑ *Request-reply*
 - ▶ A execução do serviço e sua resposta serve como confirmação que a requisição foi recebida corretamente no servidor
- ❑ *Request-reply-ack-reply*
 - ▶ Servidor recebe confirmação de que o cliente recebeu a resposta (*reply*)



Leituras adicionais

- ❑ Coulouris, G; Dollimore, J; Kindberg, T. *Distributed Systems: Concepts and Design* (3th edition), Addison Wesley, 2001
 - ▶ Capítulo 4