Sistemas Computacionais Distribuídos

Prof. Marcos José Santana

SSC-ICMC-USP

São Carlos, 2008

Grupo de Sistemas Distribuídos e Programação Concorrente

Departamento de Sistemas de Computação - SSC

Sistemas Computacionais Distribuídos

4a. Aula

Comunicação entre Processos

Conteúdo

- Como mapear estruturas de dados em mensagens
- Operações Send e Receive
- Comunicação Síncrona e Assíncrona
- Endereço de Destino de Mensagem
- Confiabilidade
- Comunicação Cliente / Servidor
- Possíveis falhas na comunicação
- Comunicação em Grupo

Comunicação entre Processos

- **Porquê é necessária:**
  - **troca de informações;**
  - **sincronismo;**
  - **cooperação.**

Como mapear estruturas de dados em mensagens

- mensagens ⇨ sequencial;
- estruturas de dados ⇨ complexidade variável;
- estrutura de dados ⇨ convertida antes do envio e reconstruída após recepção;
- pode haver conflito na representação de dados;
- valores são convertidos para uma representação comum, transmitidos e reconstruídos na recepção.

Como mapear estruturas de dados em mensagens

- Ambientes heterogêneos ⇨ conversão é necessária.
- Com comunicação orientada a conexão (TCP) ⇨ pode haver negociação.
- Alternativa ⇨ enviar na forma nativa uma informação ⇨ destino converte se necessário.
- Padrão SUN ⇨ XDR
  Padrão Xerox ⇨ Courier

Como mapear estruturas de dados em mensagens

MARSHALLING

- **Processo de montagem dos dados para transmissão.**

UNMARSHALLING

- **Processo inverso.**

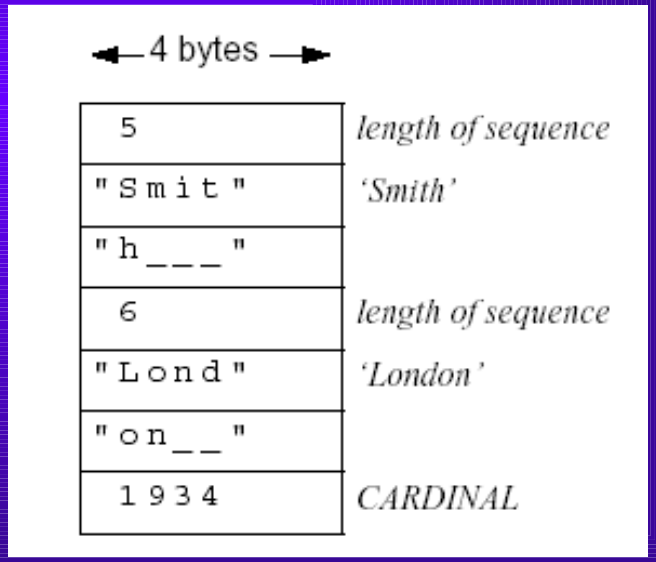
- **Na montagem: dados estruturados ⇨ dados sequenciais ⇨ representação que possa ser entendida (XDR, p. ex.).**
- **Na reconstrução: dados são convertidos da representação comum para a da máquina e as estruturas recuperadas.**
- **Marshalling ⇨ explicitamente pelo programa ou gerado automaticamente.**
- **XDR e Courier ⇨ permitem Marshalling e unMarshalling automáticos.**

Como mapear estruturas de dados em mensagens

Exemplos

Mensagem ‘Smith’, ‘London’, 1934

- **SUN XDR**



Notação: Em Courier

```
Person: TYPE = RECORD [
        name, place : SEQUENCE OF CHAR;
        year : CARDINAL ]
```

Como mapear estruturas de dados em mensagens

- Marshalling explícito (na mão!)
- converter dados para ASCII.

Ex.: 5 Smith 6 London 1934

Em C:

```
char *name = "Smith", place = "London";
int year = 1934;
sprintf (message, "%d %s %d %s %d",
      strlen(name), name, strlen(place), place, year);
```

Funciona, mas há perda de bandwidth!

Operações Send e Receive

- Send (dest, message)
- Receive (source, message)

Comunicação Síncrona e Assíncrona

- Uso de filas ⇨ envio insere mensagem.
  recepção retira mensagem.
- Modo síncrono ⇨ fonte e destino sincronizam-se em cada mensagem.
  - operações bloqueantes.
- Modo Assíncrono ⇨ fonte não é bloqueada ao enviar mensagem (uso de *buffer* local).
- Uso de *timeouts* ⇨ libera uma operação bloqueante após um certo tempo de espera.

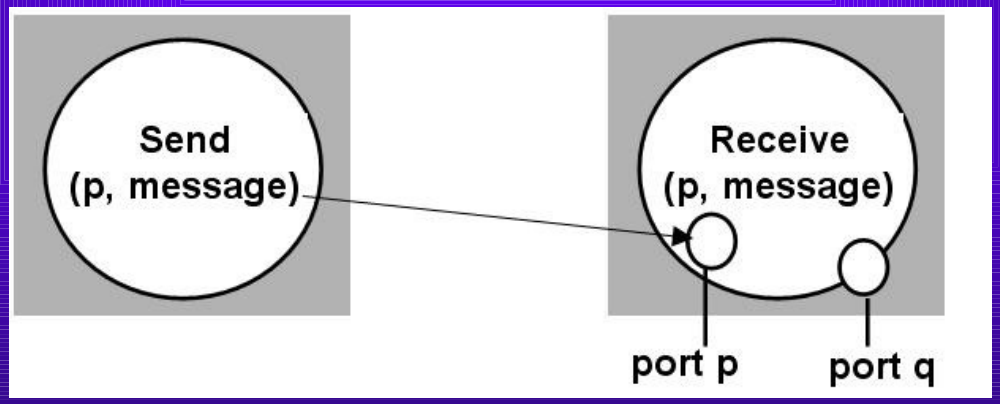
Endereço de Destino de Mensagem

- Deve ser conhecido pelo processo fonte (clientes).
- Na Internet ⇨ nº de um porto associado ao processo destino + endereço internet da máquina.
- Para ser transparente ⇨ serviço de nomes.

Endereço de Destino de Mensagem

- Para onde enviar as Mensagens
  - portos ⇨ pontos de acesso aos processos.
  - grupos de destinos ⇨ uma mensagem para vários pontos.

Ex.:



  - portos podem ser fixos ou criados dinamicamente.

Confiabilidade

- Não confiável ⇨ transmissão sem confirmação.

  Ex.: UDP

- Processo deve ter o seu mecanismo para checar se a mensagem foi entregue.

- Mensagens podem ser:
  - perdidas;
  - duplicadas;
  - entregues fora da ordem;
  - atrasadas.

Confiabilidade

- No caso de LANs ⇨ confiabilidade é maior, mas ainda podem ocorrer erros.

  ex.: checksum, buffer cheio, etc.

- Serviço confiável pode ser construído sobre um serviço não confiável através de mensagens de confirmação.

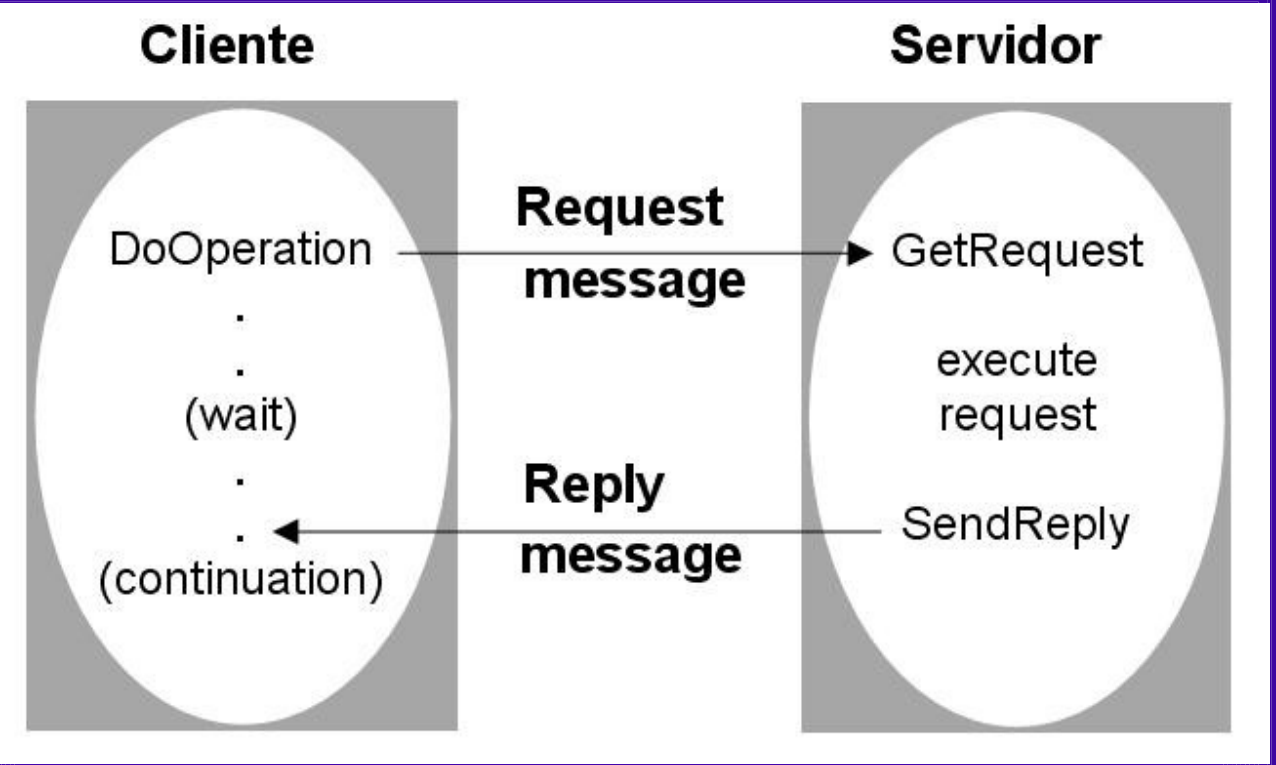
- Quanto mais confiável ⇨ há mais sobrecarga.

Comunicação Cliente / Servidor

- **Request - Reply ⇨**

messageType	(Request,Reply)
requestId	CARDINAL
procedureId	CARDINAL
arguments	(Lista Sequencial)

Comunicação Cliente / Servidor



DoOperation ⇨ usa um Send e um Receive.

GetRequest ⇨ usa um Receive e um Send.

Possíveis falhas na comunicação

- Perda da mensagem enviada.
- Perda da resposta enviada.
- Duplicação de mensagem.
- Time-out.

Comunicação em Grupo

- ***Multicast Message***

- 1 processo ⇨ grupo de processos.

- bom para:

  - tolerância a falhas baseada em serviços replicados;
  - localização de objetos em serviços distribuídos;
  - melhorar desempenho através de dados replicados;
  - atualizações múltiplas.

Comunicação em Grupo

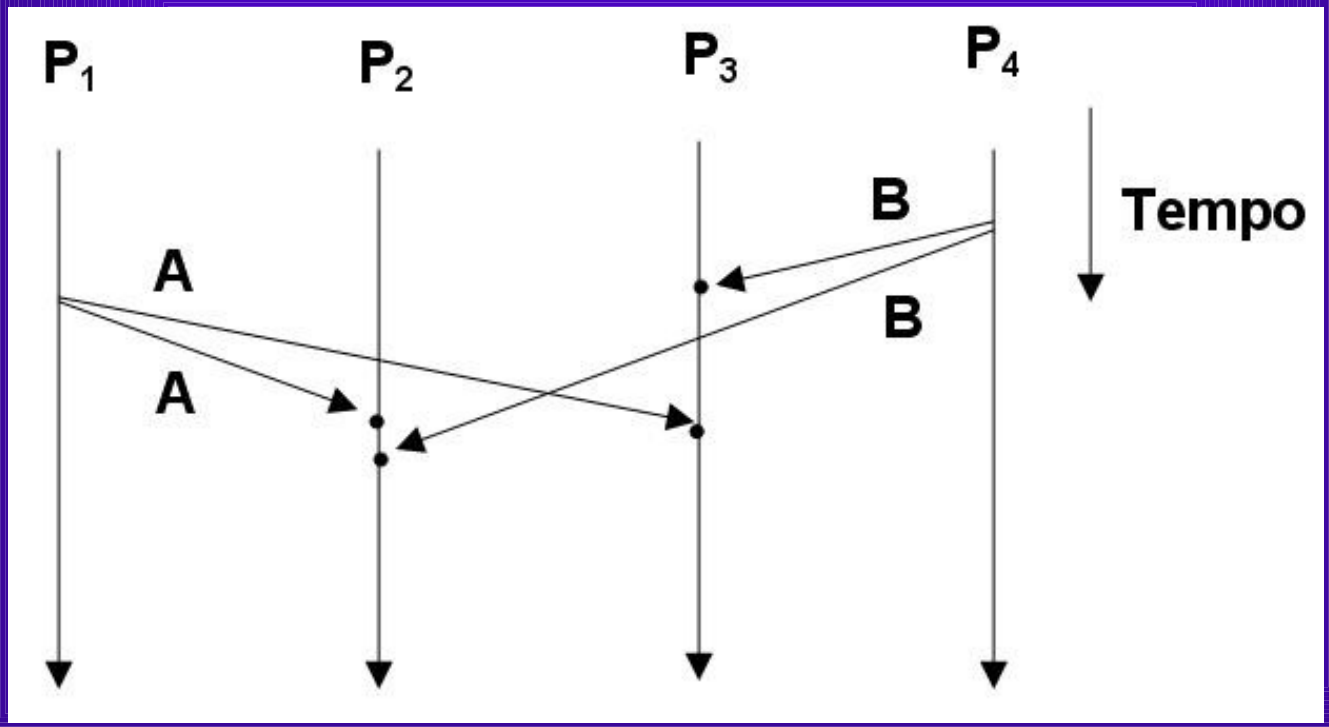
- ***Comunicação atômica***

  - **msg multicast atômica ⇨ recebida por todos os processos ou por nenhum.**

  - **msg multicast confiável ⇨ tenta enviar a todos mas não garante sucesso total.**

Comunicação em Grupo

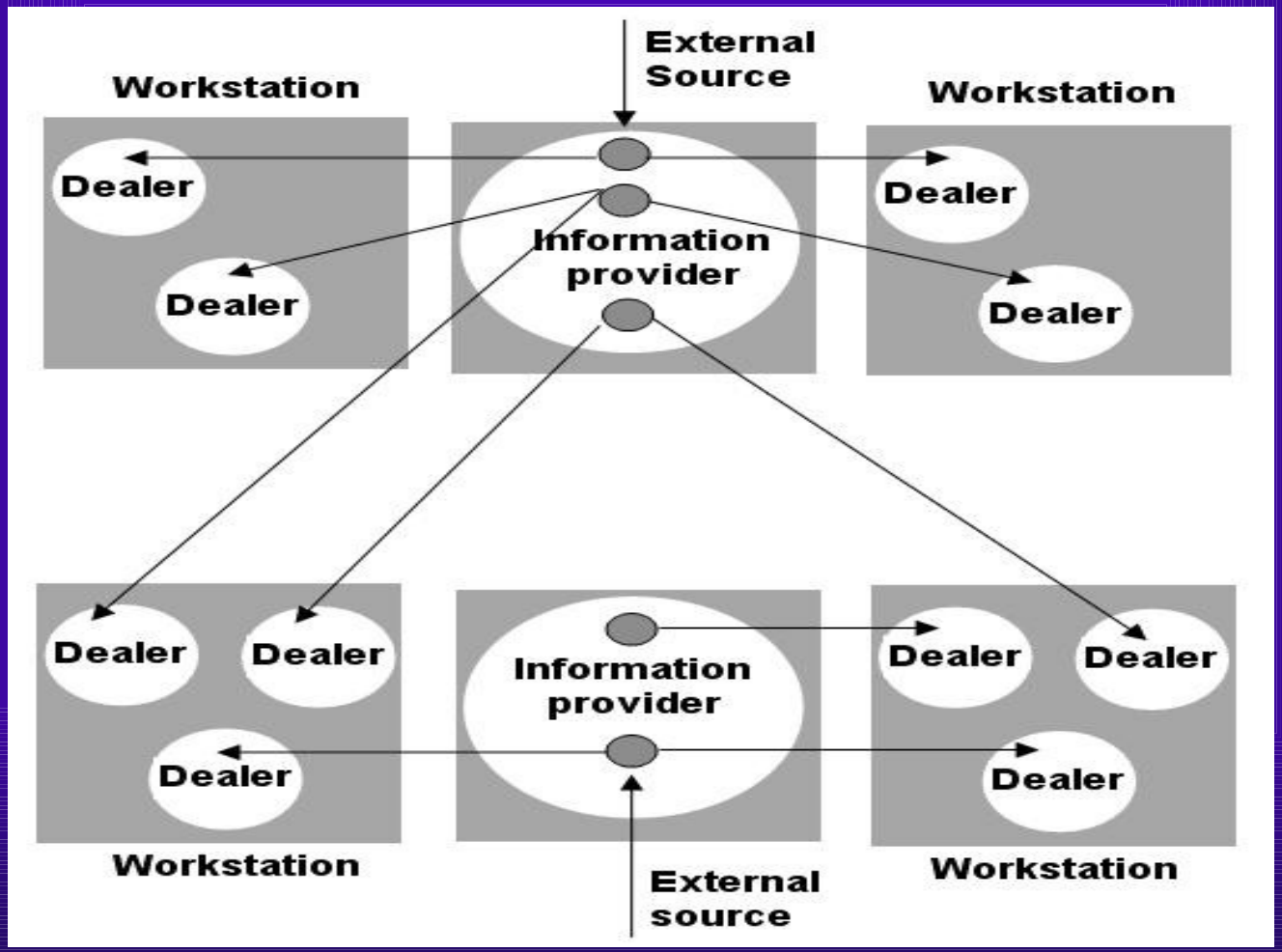
- ***Ordenação de mensagens***



⇨ **Número de seqüência.**

Comunicação em Grupo

- ***Exemplo de Aplicação***



Fim