Modelos de Programação Distribuída

CAPÍTULO II

Ricardo Costa – rcosta@estg.ipp.pt

Modelos de Programação Distribuída

Sistema Distribuído

- Conjunto de computadores ligados em rede, com software que permita a partilha de recursos e a coordenação de actividades, oferecendo idealmente um sistema integrado.

Computação Distribuída

- a computação é dividida em várias unidades que executam concorrentemente em diferentes elementos de computação. Estes podem ser diferentes processadores em diferentes máquinas, diferentes processadores na mesma máquina, ou diferentes cores no mesmo processador.

Modelos de Programação Distribuída

Um sistemas distribuído é o resultado da interação de vários componentes que atravessam toda a stack de computação desde o Hardware até ao Software.

Hardware

- Computador mais infraestrutura de rede;
- O hardware é gerido pelo Sistema Operativo (SO) que fornece os serviços para:
  - comunicação entre processos (IPC – Inter Process communication)
  - escalonamento e gestão de processos
  - gestão de recursos, como o "file system" e periféricos

Hardware + SO = Plataforma

Modelos de Programação Distribuída

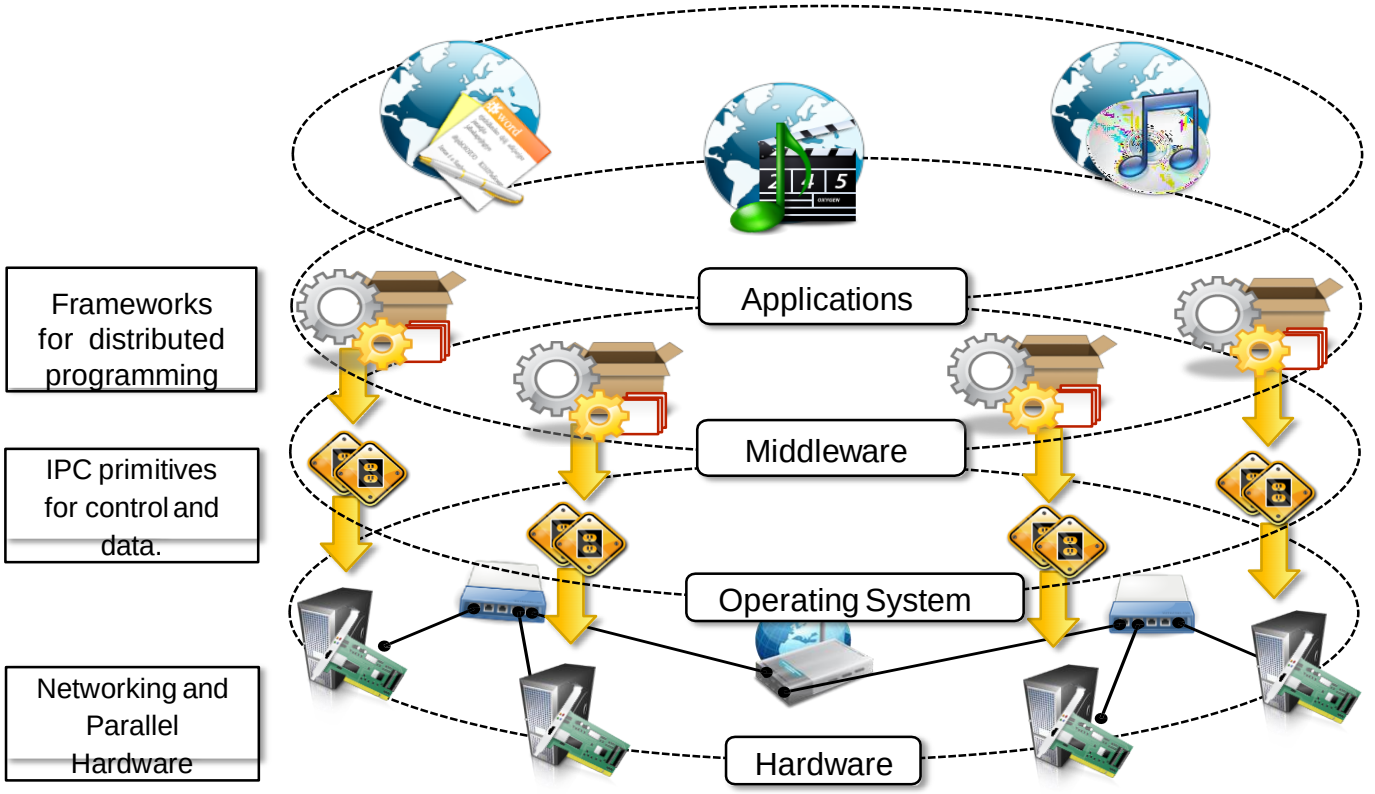
Ao nível do SO, a comunicação entre processos é implementada usando protocolos:

- TCP/IP (Transmission Control Protocol / Internet Protocol);
- UDP (User DatagramProtocol);
- Outros.

Acima do SO, existe o *middleware* que constrói uma camada de abstração que permitirá às aplicações abstraírem-se da heterogeneidade do hardware e do próprio sistema operativo.

O *middleware* fornece uma interface uniforme para o desenvolvimento de aplicações.

Modelos de Programação Distribuída



From: Mastering Cloud Computing - Foundations and Applications Programming

Modelos de Programação Distribuída

A comunicação entre processos (IPC) é usada para transferir dados entre os processos e para coordenar a sua atividade.

Modelos de IPC mais importantes:

- Memória partilhada
- Comunicação por mensagens

Modelos de Programação Distribuída

Sistemas de Memória Partilhada:

- Os processos acedem a um único espaço de endereçamento;
- Comunicação através de variáveis partilhadas;
- Sincronização feita pelas técnicas clássicas da programação concorrente (ex. Semáforos ou Monitores).

Modelos de Programação Distribuída

Sistemas de Memória Distribuída:

- Vários espaços de endereçamento disjuntos (cada processador tem a sua própria memória local
- Comunicação por mensagens (através de uma canal de comunicação)
- Comunicação e sincronização integradas num único conceito

O programador utiliza os mecanismos de comunicação por mensagens sem se preocupar com a forma como é feito o armazenamento e a transferência dos dados.

Modelos de Comunicação por Mensagem

As várias formas de comunicação por mensagens distinguem-se por:

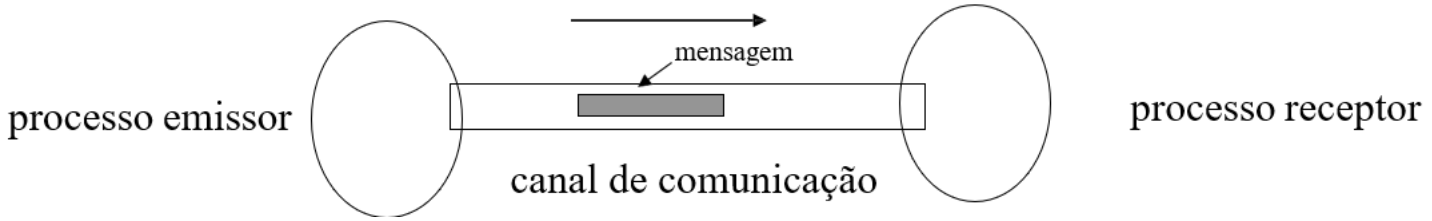
1 - Tipo de sincronização (ou interação):

- Comunicação síncrona
- Comunicação assíncrona
- Invocação remota de procedimentos

2 - Forma como são especificados os vários intervenientes no processo:

- Identificação dos processos
- Criação dos processos (dinâmica ou estática)
- Comunicação bidirecional ou unidirecional

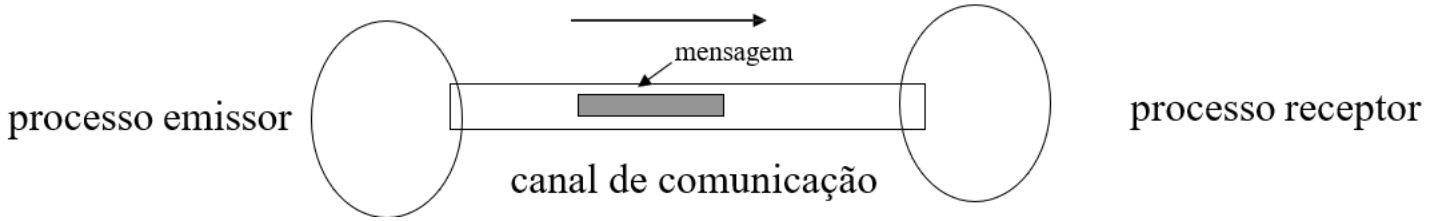
Tipo de sincronização (ou interação)



Três tipos de interação:

- Comunicação síncrona
  - O envio de uma mensagem é uma operação atómica que requer a participação de dois processos (emissor e recetor).
  - Se o emissor está pronto a enviar a mensagem mas o recetor não a pode receber, o emissor bloqueia;
  - Se o recetor está pronto a receber a mensagem mas o emissor não a envia, o recetor bloqueia.

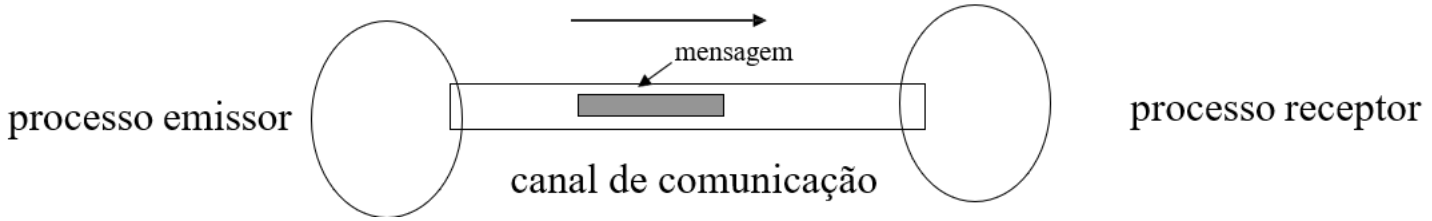
Tipo de sincronização (ou interação)



Três tipos de interação:

- Comunicação assíncrona
  - O emissor pode enviar a mensagem e continuar a executar sem bloquear, independentemente do estado do processo recetor;
  - O recetor pode estar a executar quaisquer outras instruções, e mais tarde testar se tem mensagens a receber. Quando aceita a mensagem não sabe o que se passa no emissor (pode até já ter terminado).

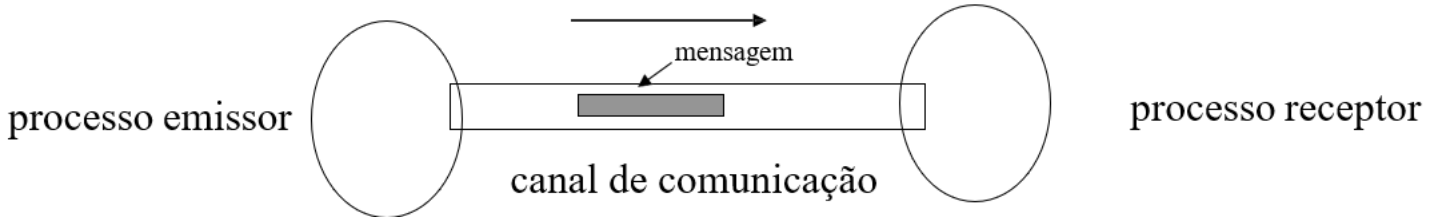
Tipo de sincronização (ou interação)



Três tipos de interação:

- Comunicação síncrona versus assíncrona (telefonar versus enviar uma mensagem)
  - Síncrona:
    - conceito de mais baixo nível (mais eficiente)
  - Assíncrona:
    - permite um maior grau de concorrência
    - exige que o sistema de execução faça a gestão e o armazenamento das mensagens (um buffer de memória tem que estar preparado para armazenar um número de mensagens potencialmente ilimitado.

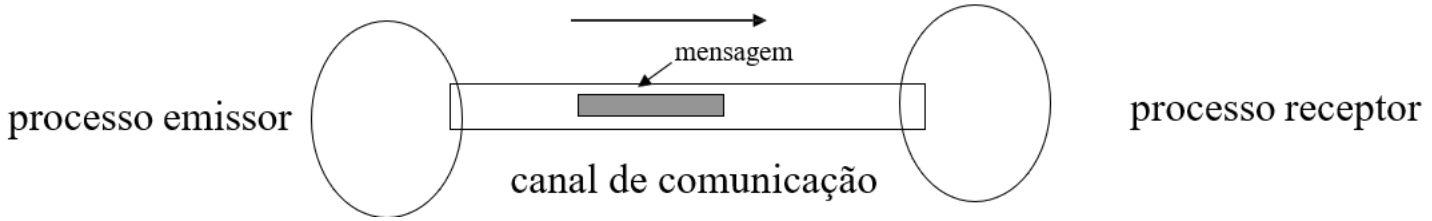
Tipo de sincronização (ou interação)



Três tipos de interação:

- Invocação remota de procedimentos – RPC (Remote Procedure Call)
  - A comunicação entre dois processos diz-se uma chamada de procedimento remoto quando:
    - Um processo (o emissor) envia um mensagem;
    - O processo recetor produz uma resposta à mensagem; e
    - O emissor permanece suspenso até à receção da resposta.

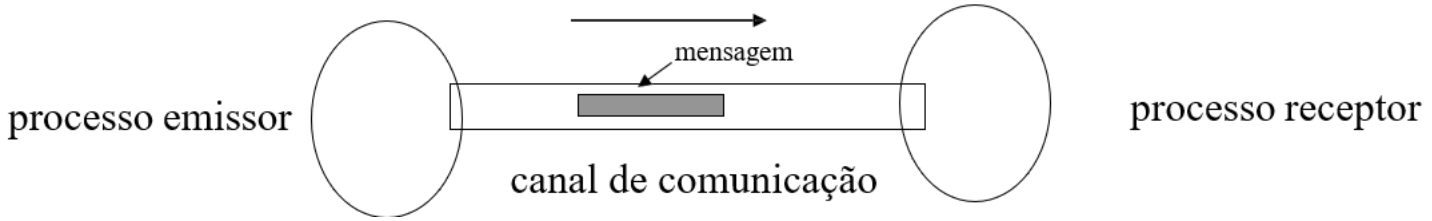
Tipo de sincronização (ou interação)



Três tipos de interação:

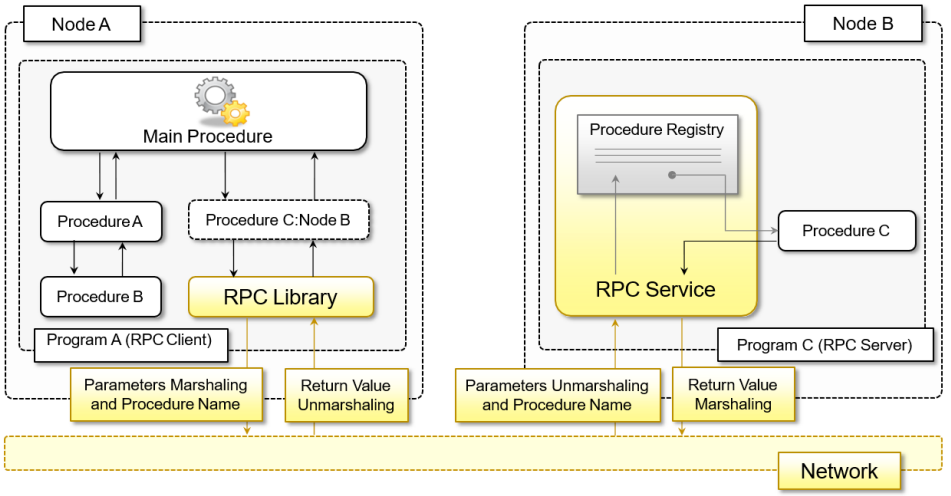
- Invocação remota de procedimentos – RPC (Remote Procedure Call)
  - Do ponto de vista do processo emissor (cliente) este mecanismo funciona como a chamada de um procedimento.
  - O procedimento não vai ser executado no próprio processo mas sim no recetor (servidor).
  - Os dados comunicados na mensagem, funcionam como parâmetros do tipo valor.
  - A resposta do recetor pode ter a forma de parâmetros resultado.
  - O sistema de execução é responsável por encapsular sob a forma de mensagem qual o procedimento a executar e os respetivos parâmetros. A pós a execução do procedimento o resultado é uma mensagem do processo remoto para o cliente.
  - O estado do procedimento não perdura entre invocações.

Tipo de sincronização (ou interação)

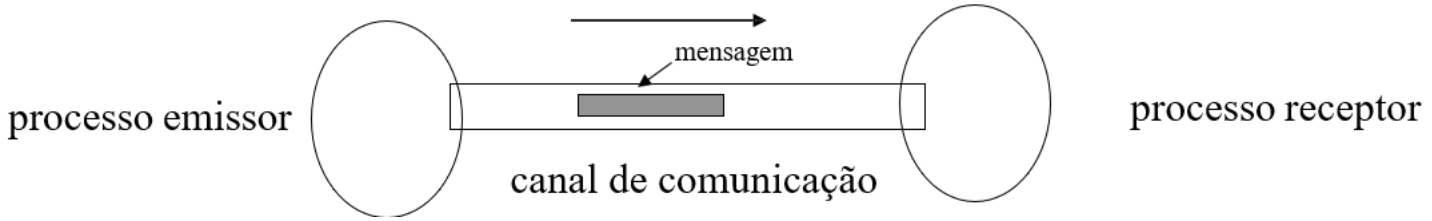


Três tipos de interação:

- Invocação remota de procedimentos – RPC (Remote Procedure Call)



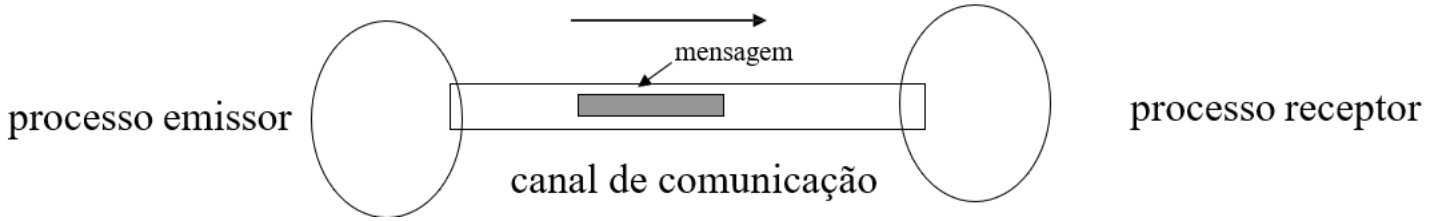
Tipo de sincronização (ou interação)



Três tipos de interação:

- Invocação remota de procedimentos – Variante do RPC (callback)
  - Emissor:
    - Após o envio da mensagem, o emissor prossegue a execução até que precise do resultado (permite maior concorrência);
    - Se, nesse ponto, a resposta ainda não estiver disponível, o emissor é suspenso.

Tipo de sincronização (ou interação)



Três tipos de interação:

- Invocação remota de procedimentos – Variante do RPC (callback)
  - Recetor:
    - Quando um processo executa uma instrução de aceitação de mensagem, é suspenso até à chegada da mesma;
    - Pode ocorrer que o recetor pretenda:
      - escolher uma de entre um conjunto de mensagens possíveis;
      - estabelecer condições para a receção de uma mensagem.
    - Para isso é necessário que exista uma instrução em que o recetor:
      - Seleciona uma de um conjunto de mensagens alternativas;
      - Cada uma das quais pode ter associada uma condição para a aceitação da mensagem.

Forma como são especificados os vários intervenientes no processo

Identificação dos processos

1. Sistema em que todos os processos têm um nome único

   - O comando de envio pode nomear directamente o processo receptor:

     ENVIA <mensagem> PARA <nome do processo>

   - Simetricamente no recetor

     ESPERA <mensagem> DE <nome do processo> (*)

     (*) requer que o receptor saiba o nome de todos os processos passíveis de lhe enviar uma mensagem

     - Se o receptor apenas estiver interessado em receber determinada mensagem, não importando quem é o emissor:

     ESPERA <mensagem> // emissor é anónimo o receptor não

Forma como são especificados os vários intervenientes no processo

Identificação dos processos

2. Quando não é apropriado um sistema de nomes únicos para todos os processos, definem-se entidades intermédias, caixas de correio (ou canais) conhecidas por ambos os intervenientes na comunicação

ENVIA <mensagem> PARA <caixa de correio>

ESPERA <mensagem> DE <caixa de correio>

Forma como são especificados os vários intervenientes no processo

Identificação dos processos

2. Quando não é apropriado um sistema de nomes únicos para todos os processos, definem-se entidades intermédias, caixas de correio (ou canais) conhecidas por ambos os intervenientes na comunicação

   - Uma caixa de correio pode ter várias formas. Pode ser usada por:
     - por vários emissores e vários receptores
     - um emissor e vários receptores (difusão ou "broadcasting")
     - vários emissores e um receptor
     - um receptor e um emissor

   - Pode ainda ser estruturada para enviar informação em ambas as direcções ou apenas numa.

Forma como são especificados os vários intervenientes no processo

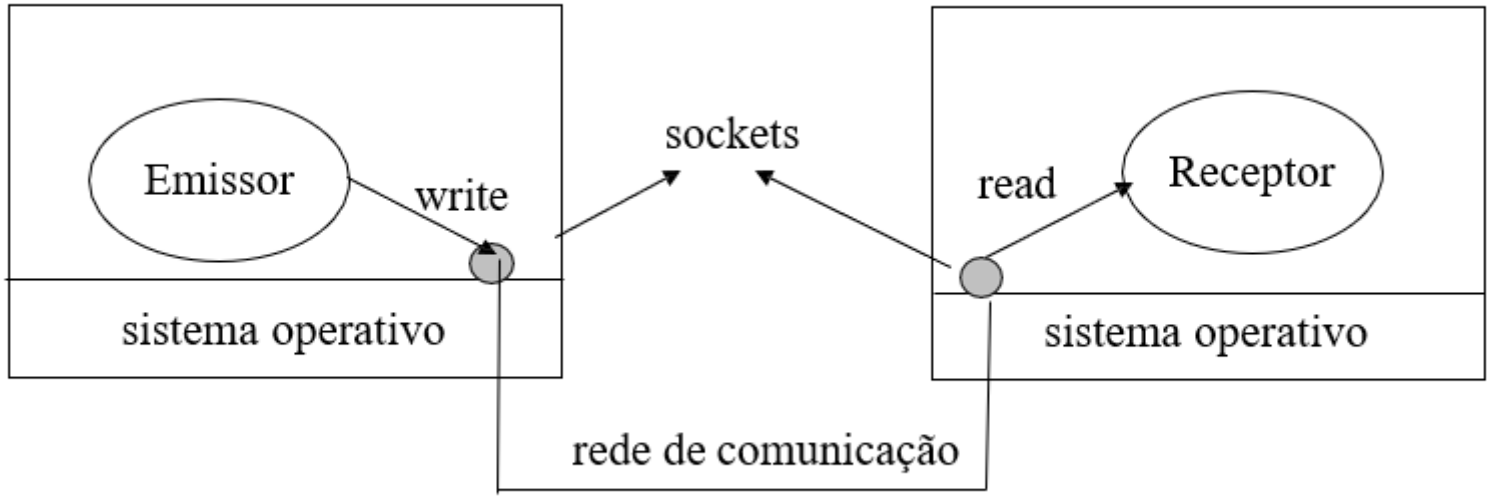
Formas de criação de processos

- Os processos de um programa distribuído podem ser criados de forma estática ou dinâmica
  - Definição estática:
    - Todos os processos são criados no início da execução.
    - A atribuição de recursos (memória, canais de comunicação, etc) é feita em tempo de compilação)
    - mais eficiente
  - Definição dinâmica:
    - Permite maior flexibilidade:
      - O sistema ajusta-se às necessidades da aplicação ao longo do tempo
      - Permite mecanismos de balanceamento de carga ("load balancing")
    - A criação estática de processos é apropriada para sistemas dedicados onde a configuração do sistema é conhecida à partida.
    - Exemplos:
      - "embedded systems"
      - sistemas de monitorização médica

Exemplos

a) Comunicação por mensagens através de Sockets TCP (em Java)

- Socket – interface de um canal de comunicação entre dois processos
- Um par de processos comunica através de uma par de sockets



Exemplos

a) Comunicação por mensagens através de Sockets TCP (em Java)

- Paradigma de comunicação (análogo a usar um descritor de um ficheiro)
  - criação (“open”) do socket
  - ler e escrever (“receive” , “send”)
  - destruição (“close”) do socket
- O endereço de um socket é especificado por:
  - endereço internet:
    - IP da máquina onde está o processo com que queremos comunicar
  - nº de um porto:
    - um porto é representado por um inteiro >1024 que identifica os vários serviços em execução numa mesma máquina, 0-1023 reservados a utilizadores com privilégios root ou superuser)
  - Exemplo: 146.75.4.30/1234

Exemplos

a) Comunicação por mensagens através de Sockets TCP (em Java)

- A classe Socket
  - Permite criar sockets que comunicam através do protocolo TCP (Transmission Control Protocol) usando uma stream de bytes.
  - Um dos sockets (o servidor) aguarda por um pedido de ligação enquanto o outro (o cliente) solicita a ligação.
  - Após ser estabelecida a ligação entre os dois sockets, estes podem ser usados para transmitir dados nos dois sentidos

Exemplos

a) Comunicação por mensagens através de Sockets TCP (em Java)

- A classe Socket
  - 1) Criar um socket
    - Se o processo é um cliente:

```
import java.net.*;
import java.io.*;
Socket meuCliente = null;
try {
   meuCliente = new Socket (“host”, portNumber);
} catch (IOException e){
   System.out.println( e.getMessage());
}
```

Exemplos

a) Comunicação por mensagens através de Sockets TCP (em Java)

- A classe Socket
  - 1) Criar um socket
    - Se o processo é um servidor:

```
...
ServerSocket meuServidor = null;
try {
   meuServidor = new ServerSocket (portNumber);
} catch (IOException e) { System.out.println( e.getMessage()); }

Socket sServidor = null
try {
   sServidor = meuServidor.accept();*
} catch (IOException e) { System.out.println( e.getMessage()); }
```

* método da classe ServerSocket: public Socket accept() throws IOException

Exemplos

a) .1 - Percorrer todos os portos de um máquina remota ...

```
public static void scan(String host) {

    for (int port = 0; port<65536; port++) {
        try {
            Socket s = new Socket(host , port);
            System.out.println("Existe um servidor no porto " + port + " da máquina " + host );
            s .close();
        } catch (IOException e) {
            System.out.println("A máquina " + host + não está à escuta no porto " + port);
        }
    }
}
```

Exemplos

a) Comunicação por mensagens através de Sockets TCP (em Java)

- A classe Socket
  - 2) Criar uma OutputStream
    - No cliente

```
...
PrintWriter os = null;
try {
   os = new PrintWriter(a) ( meuCliente.getOutputStream()(b) , true);
   os.println ( "Olá, eu sou o cliente" );
} catch (IOException e) {
   System.out.println( e.getMessage());
}
```

(a) PrintWriter (OutputStream out, boolean autoFlush); – converte caracteres em bytes

(b) OutputStream getOutputStream() throws IOException; – método da classe Socket

Exemplos

a) Comunicação por mensagens através de Sockets TCP (em Java)

- A classe Socket
  - 2) Criar uma OutputStream
    - No servidor

```
...
os = new PrintWriter (sServidor.getOutputStream(), true);
...
```

  - 3) Criar uma InputStream
    - No cliente

```
BufferedReader is = null;
try {
   is = new BufferedReader (new InputStreamReader (meuCliente.getInputStream()) );
   is.readLine();
} catch ( ....
```

Exemplos

a) Comunicação por mensagens através de Sockets TCP (em Java)

- A classe Socket
  - 3) Criar uma InputStream
    - No servidor

```
...
is = new BufferedReader (new InputStreamReader (sServidor.getInputStream()));
...
```

  - 4) Fechar um Socket

1º fechar as streams

```
try {
   is.close();
   os.close();
  // fechar os sockets
  meuCliente.close(); // no cliente
  sServidor.close();     // no servidor
} catch (IOException e) {...} ...
```

Exemplos

b) Sistemas de Objetos distribuídos

- Os Sistemas de Objetos Distribuídos implementam o modelo de objectos locais usando RPC.
  - Cada processo regista um conjunto de interfaces que podem ser acedidas remotamente.
  - Os processos clientes podem aceder a um ponteiro para essas interfaces e invocar os métodos disponíveis através delas.
  - Os métodos remotos executam no contexto de uma instância de um objeto cuja existência pode perdurar independentemente de existência de "requests" dando origem a "active objects" que têm uma thread de execução própria.
  - Ex.s CORBA, Java RMI, .NET Remoting

Exemplos

c) Web Services

- Implementam o modelo de RPC usando o protocolo HTTP.
  - Um WS é exposto como um objeto remoto num servidor web.
  - As invocações dos métodos são transformadas em pedidos (requets) HTTP.
  - Esses requests são encapsulados segundo dois protocolos principais: SOAP – Simple Object Access Protocol
  - REST – Representational State Transfer

Modelos de Comunicação por Mensagem

Comunicação ponto a ponto

- Cada mensagem é enviada de um componente para outro.
- O processo que envia a mensagem tem de conhecer o endereço do receptor.
- A comunicação pode ser síncrona ou assíncrona.
- A comunicação assíncrona implica um sistema de armazenamento de mensagens.

Modelos de Comunicação por Mensagem

Publish-and-subscribe

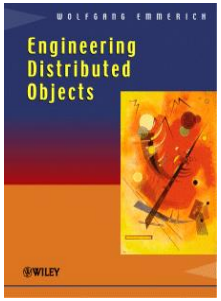
- Os processos podem desempenhar um de dois papéis:
- O publisher que permite aos outros processos subscreverem um determinado tópico ou evento.
- Quando o evento ocorre do lado do publisher, é desencadeada a criação de uma mensagem associada ao evento e que ficará disponível para todos os subscritores desse evento.
- Para os subscritores serem notificados do evento, há duas estratégias possíveis:
  - Push strategy
    - O publisher tem a responsabilidade de notificar todos os subscritores.
  - Pull strategy
    - O publisher cria a mensagem, mas são os subscritores que têm que verificar se há mensagens para um dado evento.

Modelos de Comunicação por Mensagem

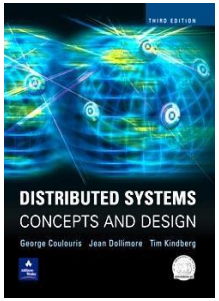
Request-reply

- Inclui todos os modelos para os quais para cada mensagem enviada a um processo existe uma resposta.

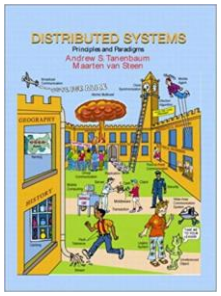
Bibliografia



From: Wolfgang Emmerich
Engineering Distributed Objects
John Wiley & Sons, Ltd 2000



From: Coulouris, Dollimore and Kindberg
Distributed Systems: Concepts and Design
Edition 4 © Addison-Wesley 2005



From: Andrew S., Tanembaum and Van Steen, Maarten
Distributed Systems: Principles and Paradigms
Edition 2 © Pearson 2013

Questões?