Capítulo II – Modelos de Programação Distribuída

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

From: M. Ben-Ari
Principles of Concurrent and Distributed Programming
Prentice Hall 1990

From: Qusay H. Mahmoud
Distributed Programming with Java
Manning Publications, 1999

From: Mastering Cloud Computing
Foundations and Applications Programming
Rajkumar Buyya, Christian Vecchiola, Thamarai Selvi
Morgan Kaufmann Publishers, 2013

Paula Prata,
Departamento de Informática da UBI
http://www.di.ubi.pt/~pprata

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 pilha (stack) de computação desde o hardware até ao Software:

Hardware: computadores mais infra-estrutura de rede;

O hardware é diretamente 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

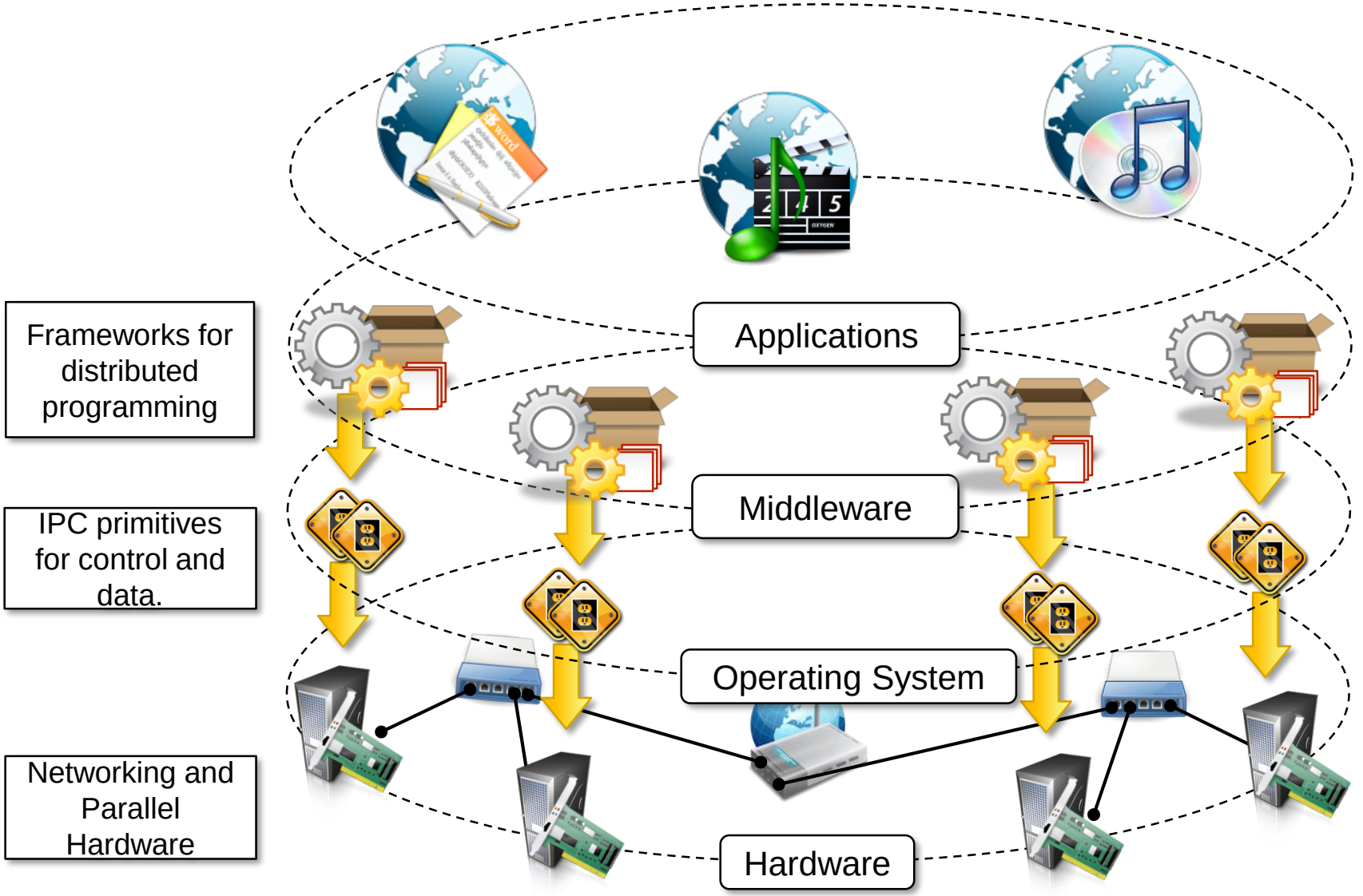
Ao nível do sistema operativo, a comunicação entre processos é implementada usando protocolos como:

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

Acima do sistema operativo, o middleware constrói uma camada que permitirá às aplicações abstraírem da heterogeneidade do hardware e do sistema operativo das plataformas.

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

Modelos de Programação Distribuída



From: Mastering Cloud Computing - Foundations and Applications Programming

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

Modelos de IPC mais importantes:

- *Memória partilhada*
- *Comunicação por mensagens*

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).

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.

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

1 - Tipo de sincronização (ou interação)

- Comunicação síncrona
- Comunicação assíncrona
- Invocação remota de procedimentos

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

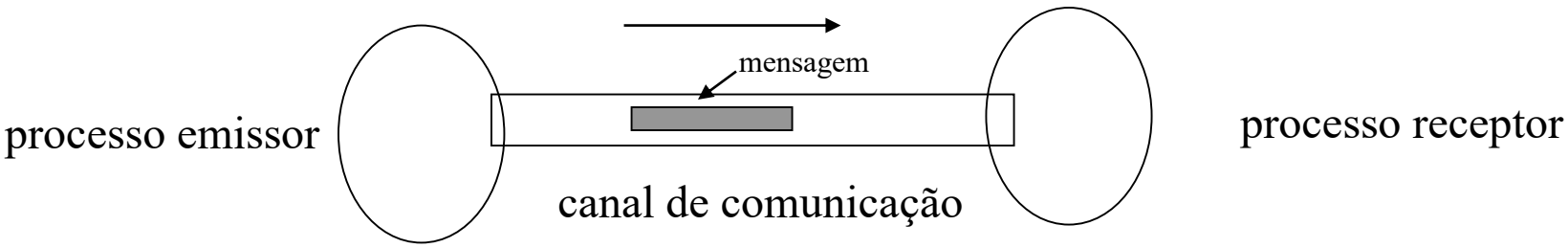
2 - Forma como são especificados os vários intervenientes no processo:

- Identificação dos processos

- Criação dos processos (dinâmica ou estática)

- Comunicação bi ou uni-direcional

1 – Modelos de comunicação por mensagens



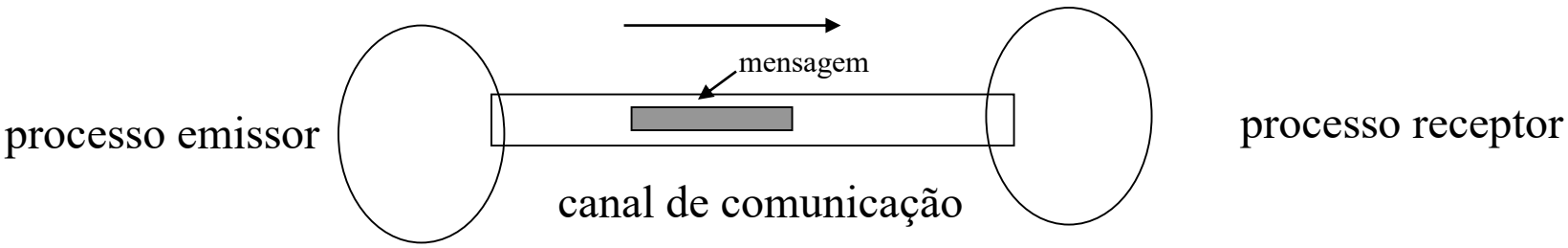
Três tipos de sincronização (ou formas de interacção):

Comunicação síncrona

O envio de uma mensagem é uma operação atómica que requer a participação de dois processos (emissor e receptor)

- Se o emissor está pronto a enviar a mensagem mas o receptor não a pode receber, o emissor bloqueia.

- Se o receptor está pronto a receber a mensagem mas o emissor não a envia, o receptor bloqueia



Comunicação síncrona (cont.)

Em resumo:

O acto de comunicação sincroniza as sequências de execução dos dois processos

o 1º processo a chegar ao ponto da comunicação espera pelo segundo

Comunicação assíncrona

- O emissor pode enviar a mensagem e continuar a executar sem bloquear, independentemente do estado do processo receptor.

- O receptor 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)

Comunicação síncrona versus assíncrona

(telefonar vs enviar uma mensagem)

c. síncrona – conceito de mais baixo nível (mais eficiente)

c. 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.

Invocação remota de procedimentos

RPC (Remote Procedure Calling)

A comunicação entre dois processos diz-se uma chamada de procedimento remoto quando,

1 – Um processo (o emissor) envia um mensagem

2 – o processo receptor produz uma resposta à mensagem e

3 – o emissor permanece suspenso até à recepção da resposta.

Invocação remota de procedimentos (cont.)

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 receptor (servidor).

Os dados comunicados na mensagem, funcionam como parâmetros do tipo valor.

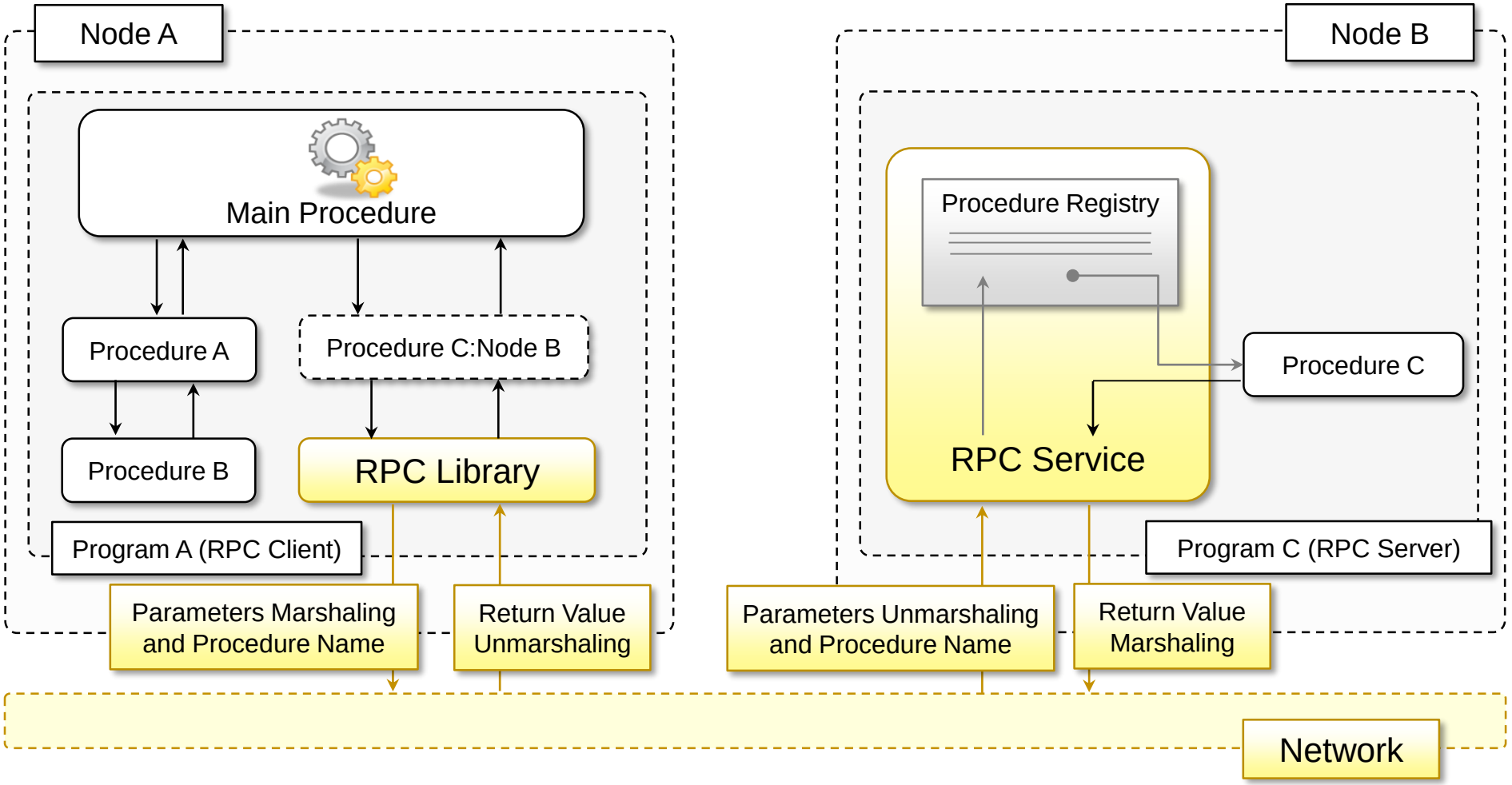
A resposta do receptor pode ter a forma de parâmetros resultado.

Invocação remota de procedimentos (cont.)

O sistema de execução é responsável por encapsular sob a forma de mensagem qual o procedimento a executar e os respectivos parâmetros. Após a execução do procedimento o resultado é uma mensagem do processo remoto para o cliente.

O estado do procedimento não perdura entre invocações.

Invocação remota de procedimentos (cont.)



Invocação remota de procedimentos

Variante do RPC

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

Invocação remota de procedimentos

Variante do RPC (cont.)

Receptor:

- Quando um processo executa uma instrução de aceitação de mensagem, é suspenso até à chegada da mesma.

Pode ocorrer que o receptor pretenda:

- escolher uma de entre um conjunto de mensagens possíveis;
- estabelecer condições para a recepção de uma mensagem;

Variante do RPC

Receptor: (cont)

Para isso é necessário que exista uma instrução em que o receptor,

- selecciona uma de um conjunto de mensagens alternativas

- cada uma das quais pode ter associada uma condição para a aceitação da mensagem

2 - Forma como são especificados os vários intervenientes no processo

Identificação dos processos

i) 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 receptor

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

Identificação dos processos

(*) 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>

// *o emissor é anónimo o receptor não*

Identificação dos processos

ii) 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>

Uma caixa de correio pode ter várias formas. Pode ser usada 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.

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*

Formas de criação de processos

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.

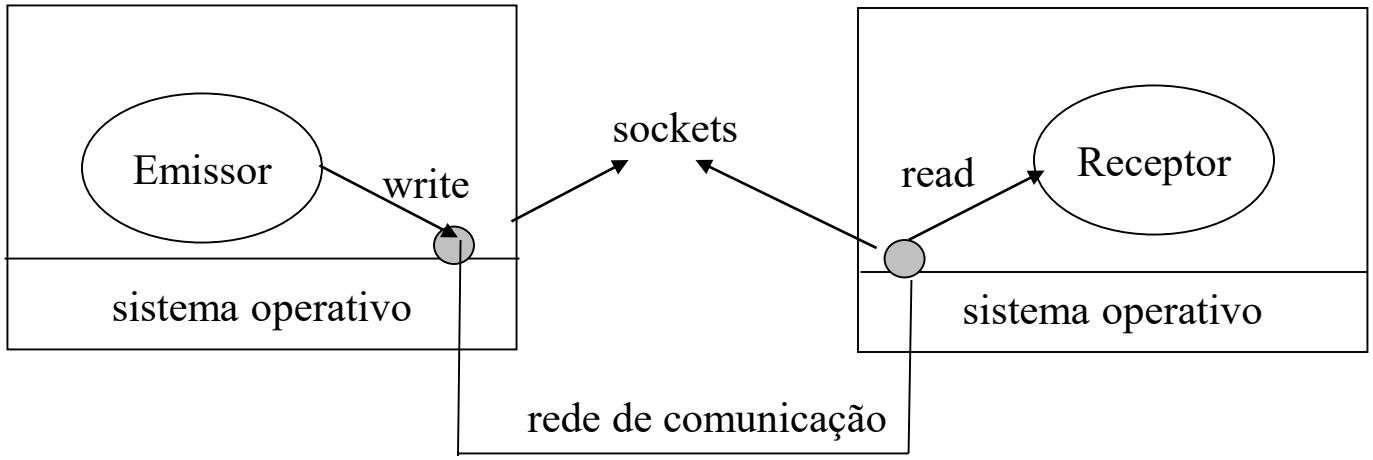
Ex.los: - “embedded systems” - sistemas de monitorização médica , ...

2 – 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 um par de sockets



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)

Ex. lo 146.75.4.30/1234

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.

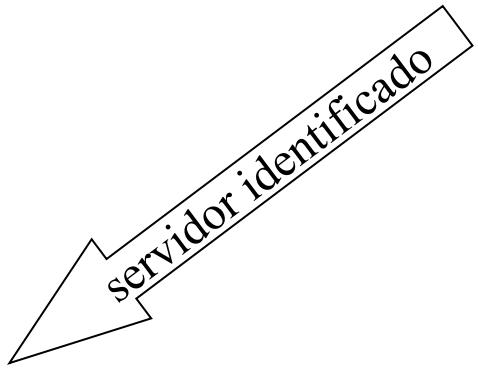
- Após ser estabelecida a ligação entre os dois sockets, estes podem ser usados para transmitir dados nos dois sentidos

2 – Exemplos:
a) Comunicação por mensagens através de Sockets (em Java)

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());
 }
```

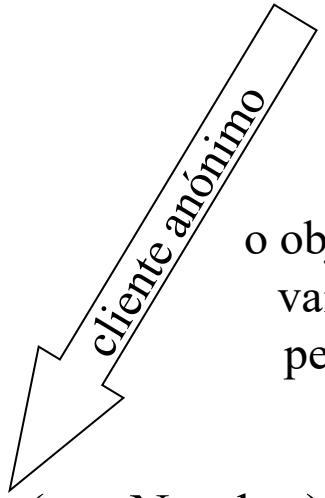


2 – Exemplos:
a) Comunicação por mensagens através de Sockets (em Java)

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());  }
```



o objecto do tipo ServerSocket vai permitir esperar por um pedido de ligação no porto especificado

quando o cliente solicita uma ligação, o método accept() devolve o endereço de uma ligação ao socket do servidor

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

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());
} ...
```

caracteres ⟶ bytes

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

(b) OutputStream getOutputStream () throws IOException; – método da 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();          <= String readLine();
} catch ( ....
```

caracteres ← bytes

2 – Exemplos:
a) Comunicação por mensagens através de Sockets (em Java)

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) {...}  ...
```

Para criar o cliente e o servidor na mesma máquina: IP: 127.0.0.1 (local host)

- permite à maquina comunicar com ela própria através do protocolo TCP/IP mesmo sem estar ligada à rede

2 – Exemplo: 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);
           }
        }
}
```

2 – Exemplos:

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

Exercício:

1- Criar um cliente e um servidor que comuniquem por Sockets TCP e Object Streams.

Suponha que o primeiro a comunicar é o cliente.

O Servidor responde.

2 – Exemplos:

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

Resolução:

…

Cliente:

```
Socket soc = new Socket( 127.0.0.1, 2000);
ObjectOuputStream oos = new ObjectOuputStream (soc.getOutputStream());

oos.writeObject (“Bom dia, como estás?”);
oos.flush();
```

…

2 – Exemplos:

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

Resolução:

…

Servidor:

```
ServerSocket  ss = new ServerSocket ( 2000);

 Socket sos =  ss.accept();
 ObjectInputStream ois = new ObjectInputStream (sos.getInputStream() );
 String s = ois.readObject ();
 System.out.println (s);
```

---

2 – Exemplos:

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

Exercício:

- Complete o exercício.

2 – Modifique o exercício de forma os dois processos possam conversar até que o servidor decida terminar a conversa. Cada processo deve ler do teclado o texto a enviar.

3 – Modifique o exercício, para que o servidor possa receber a ligação de outro cliente após terminar a conversa com o primeiro.

2 – Exemplos:

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

Exercício:

3- Classifique o modelo de comunicação quanto:

- ao tipo de sincronização;
- à forma como são especificados os intervenientes no processo (Identificação dos processos / Criação dos processos / Comunicação bi ou uni-direcional)

4 – Como será essa classificação, se usarmos sockets UDP (FP01) ?

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.

2 – Exemplos:

b) Sistemas de Objetos distribuídos

- 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, …

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

3 – Modelos de interação por mensagens:

- ***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.

3 – Modelos de interação por mensagens:

- ***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:

3 – Modelos de interação por mensagens:

- ***Publish-and-subscribe***

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.

3 – Modelos de interação por mensagens:

- ***Request-reply***

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