Capítulo II – Modelos de Programação Distribuída

From: Coulouris, Dollimore and Kindberg
Distributed Systems: Concepts and Design
Edition 3, © Addison-Wesley 2001

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: Pankaj Jalote
Fault Tolerance in Distributed Systems
Prentice-Hall 1998

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

1 – Modelos de comunicação por mensagens

Modelos clássicos de cooperação de processos:

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)

1 – Modelos de comunicação por mensagens

Comunicação por mensagens

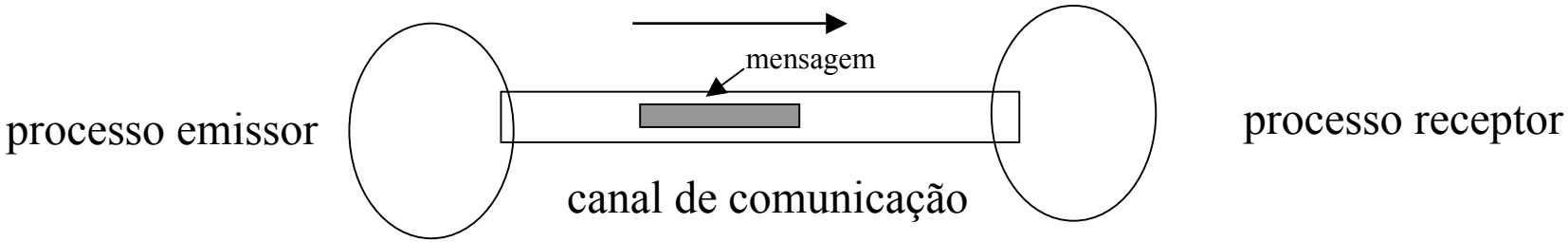
- Comunicação e sincronização integradas num único conceito

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:

- tipo de sincronização
  - comunicação síncrona
  - comunicação assíncrona
  - invocação remota de procedimentos
- forma como são especificados os vários intervenientes no processo
  - identificação dos processos
  - criação dos processos (dinâmica ou estática)
  - comunicação bi ou uni-direccional

1 – Modelos de comunicação por mensagens



Três tipos 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

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

1 – Modelos de comunicação por mensagens

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 vs assíncrona

(telefonar vs enviar uma carta)

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.

1 – Modelos de comunicação por mensagens

Invocação remota de procedimentos

RPC (Remote Procedure Calling)

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

1 – o receptor produz uma resposta à mensagem e

2 – o emissor permanece suspenso até à recepção dessa resposta

Do ponto de vista do processo emissor (cliente) este mecanismo funciona como a chamada de um procedimento,

esse 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

1 – Modelos de comunicação por mensagens

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

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

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 receptor

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

1 – Modelos de comunicação por mensagens

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*

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>

1 – Modelos de comunicação por mensagens

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.

Ex.los

OCCAM – comunicação síncrona através de canais dedicados entre pares de processos

Ada – comunicação síncrona (RPC) em que um processo comunica com outro conhecendo o seu nome mas não divulgando a própria identidade

Linda – Comunicação assíncrona com difusão de mensagens que não contêm a identificação dos processos

1 – Modelos de comunicação por mensagens

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")

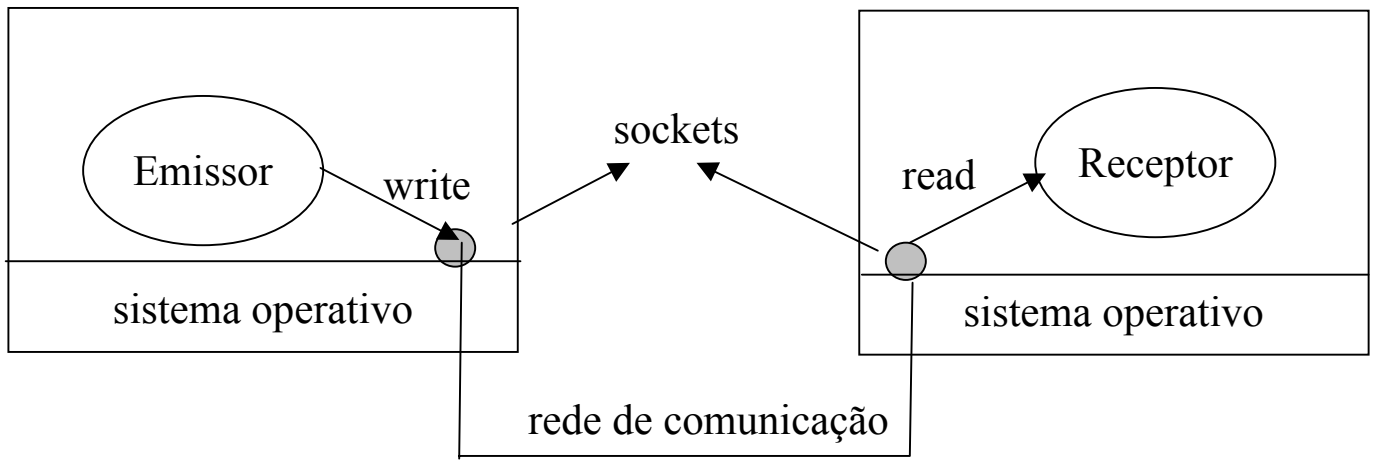
A criação estática de processos é apropriada para sistemas dedicados onde a configuração do sistema é conhecida à partida

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

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

Socket – interface de um canal de comunicação entre dois processos

Um par de processos comunica através de uma par de sockets



Paradigma de comunicação (*análogo a usar um descritor de um ficheiro*)

- criação (“open”) do socket
- ler e escrever (“send” , “receive”)
- destruição (“close”) do socket

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

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

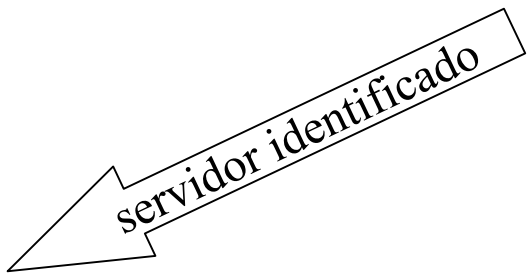
- após ser estabelecida a ligação entre os dois sockets, estes podem ser usados para transmitir dados nos dois sentidos

2 – Exemplo: 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());
 }
```

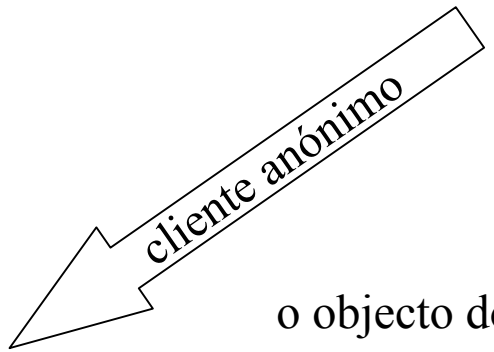


2 – Exemplo: Comunicação por mensagens através de Sockets (em Java)

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 – Exemplo: Comunicação por mensagens através de Sockets (em Java)

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 – Exemplo: Comunicação por mensagens através de Sockets (em Java)

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 ( ....
```

2 – Exemplo: 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

3 – Failure Models

Definições de “fault”, “error”, “failure”:

- Termos em português:

falha, erro, avaria (grupo de Coimbra) - adoptado neste curso

falta, erro, falha (grupo de Lisboa)

Falha (“fault”) – uma falha é uma alteração do funcionamento

de um componente (hardware ou software) do sistema

Uma falha pode ocorrer em qualquer etapa do desenvolvimento de um sistema: especificação, desenho, implementação ou durante o seu tempo de funcionamento

As Falhas de hardware são geralmente classificadas em relação à sua duração:

- falhas permanentes (“permanent faults”) - resultam de um defeito físico irreversível, permanecem indefinidamente até ser reparada.

3 – Failure Models

• falhas intermitentes (“intermittent faults”) - falhas temporárias que ocorrem repetidamente.

• falhas transitórias (“transient faults”) – falhas temporárias que ocorrem ocasionalmente num muito curto espaço de tempo.

- são as mais frequentes e mais difíceis de detectar

- podem ser causadas por oscilações na corrente eléctrica, interferências electromagnéticas ou radiação

A crescente complexidade do hardware e software aumenta a probabilidade de erros na sua concepção e implementação, assim como a susceptibilidade do hardware a factores externos

3 – Failure Models

Uma falha pode não produzir qualquer efeito, permanecendo inactiva, ou pode dar origem a uma alteração do estado do sistema, tornando-se uma falha efectiva

O intervalo de tempo entre a ocorrência da falha e a sua activação denomina-se por latência de falha

Erro ("error") – um erro é a manifestação de uma falha

Um erro provoca a corrupção de elementos de dados (afecta o estado do sistema)

Quando, como resultado de um erro, o sistema executa erradamente uma das suas funções, i.é, o sistema avaria, o erro tornou-se efectivo

O intervalo de tempo entre a ocorrência do erro e o aparecimento da avaria correspondente denomina-se por latência do erro

Avaria ("failure") – Uma avaria é qualquer alteração do comportamento do sistema em relação ao esperado (i.é, em relação à sua especificação)

3 – Failure Models

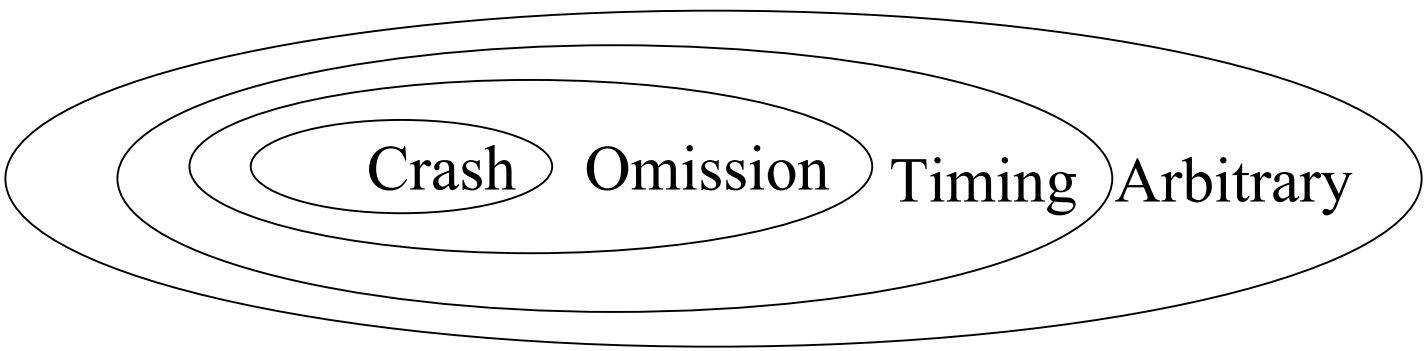
Modelos de avarias em Sistemas Distribuídos

"Crash failures" - o componente deixa de funcionar

"Omission failures" - o componente não responde a alguns dos inputs

"Timing failures" - o tempo de resposta não corresponde ao esperado

"Arbitrary failures" – o componente comporta-se de uma forma completamente arbitrária, não responde, responde no tempo indevido, responde com valores errados, ...



...