Computação Distribuída

Cap. III – Comunicação entre Processos

Licenciatura em Engenharia Informática

Universidade Lusófona

Prof. Paulo Guedes (paulo.guedes@ulusofona.pt)

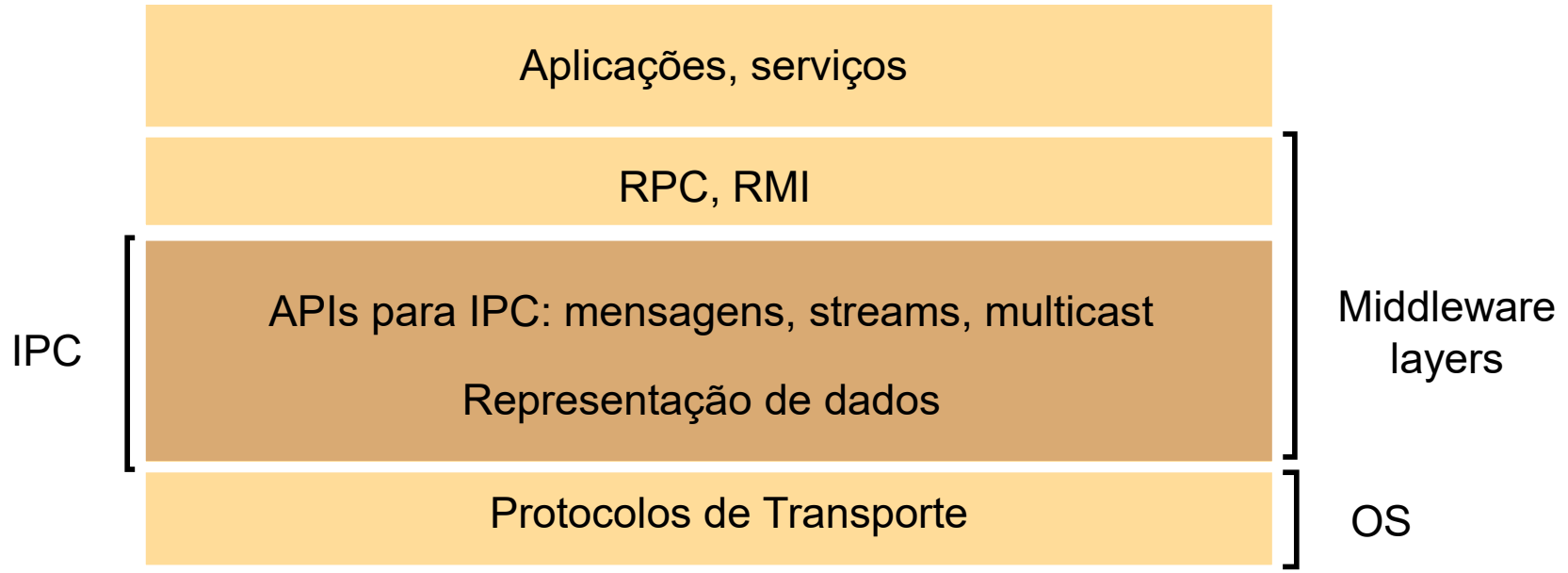
Cap. III – Comunicação entre Processos

- Modelo da comunicação distribuída
- Caracterização da interface

Sumário:

- Modelo de comunicação
- Tipos de canais, interface, semântica de envio e de receção, estrutura da informação
- Representação dos dados nas mensagens, representação normalizada dos dados, exemplo de mensagem XML
- Referências de objetos remotos

IPC - Comunicação entre Processos

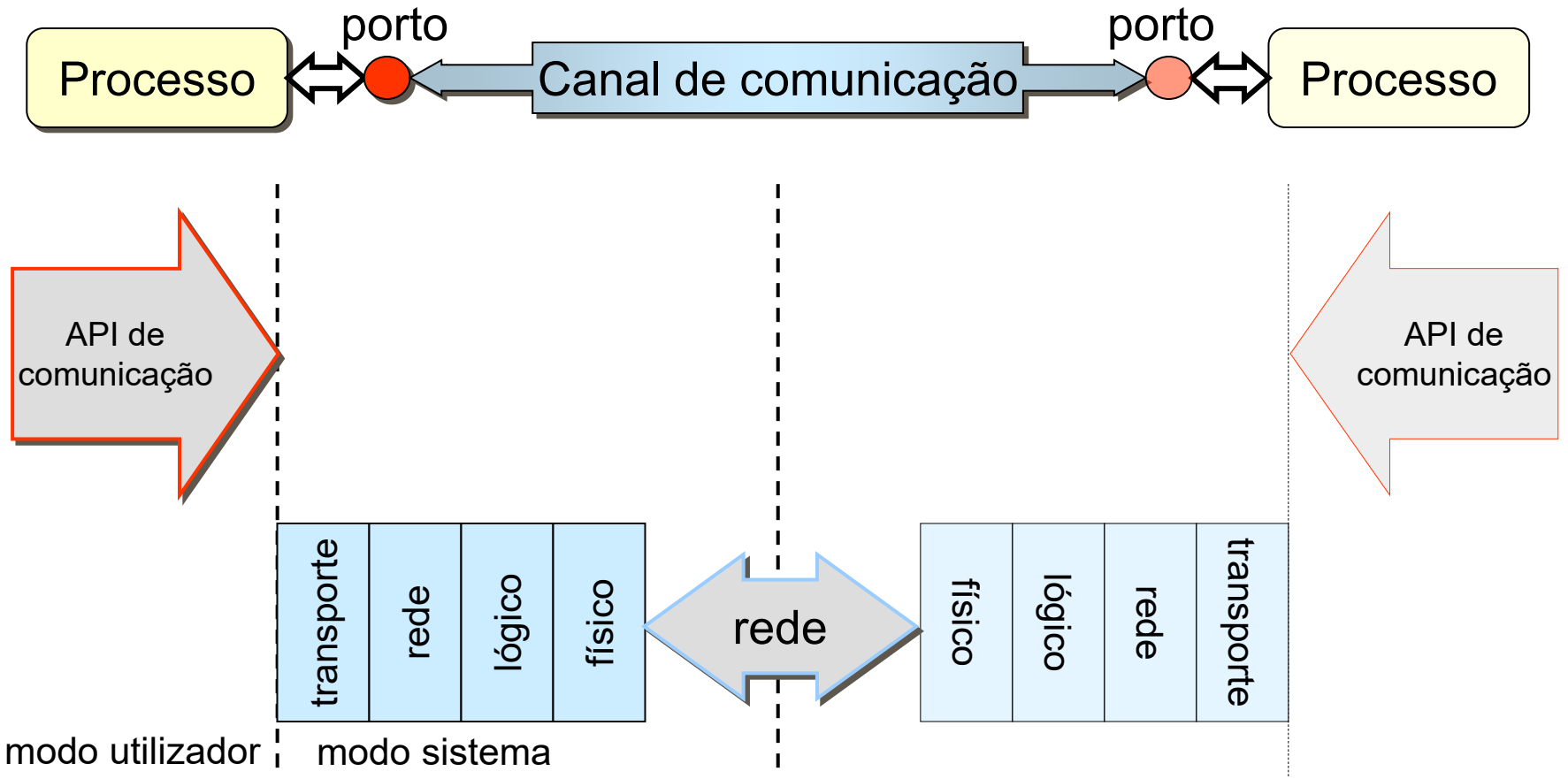


- O IPC fornece uma API (Application Programming Interface) para comunicação entre aplicações
  - Fluxos ou Mensagens
  - Protocolos de pedido/resposta
  - Sincronização, segurança, tratamento de falhas
  - Representação heterogénea de dados
- Os protocolos de transporte são implementados no SO
  - Noções de fluxo ou datagrama - TCP e UDP
  - O acesso é feito através de API de baixo nível (ex: *sockets*)

Modelo de Comunicação

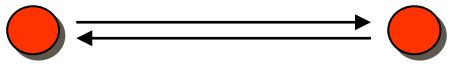
- **Comunicação**: interacção entre um processo *emissor,* que gera a informação, e um processo *receptor,* que irá tratá-la.
- **Canal**: abstracção dos mecanismos de *transporte* que suportam a transferência de informação
- **Porto**: extremidade de um canal
  - Conceptualmente, o canal pode ser visto pelos utilizadores como a associação entre dois portos

IPC: (i) modelo

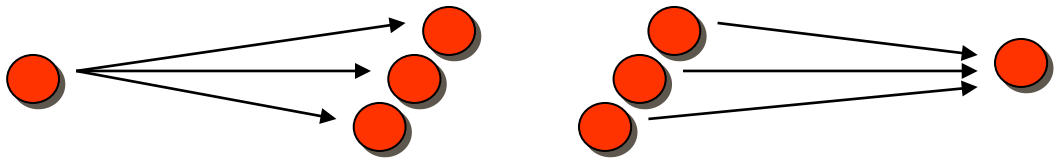

Processo
porto
Canal de comunicação
porto
Processo
API de comunicação
API de comunicação
transporte
rede
lógico
físico
rede
físico
lógico
rede
transporte
modo utilizador
modo sistema

IPC: (ii) Tipos de canais

- Com ligação
  - Normalmente serve 2 interlocutores



  - Normalmente fiável, bidireccional e garante sequencialidade
- Sem ligação
  - Normalmente serve mais de 2 interlocutores



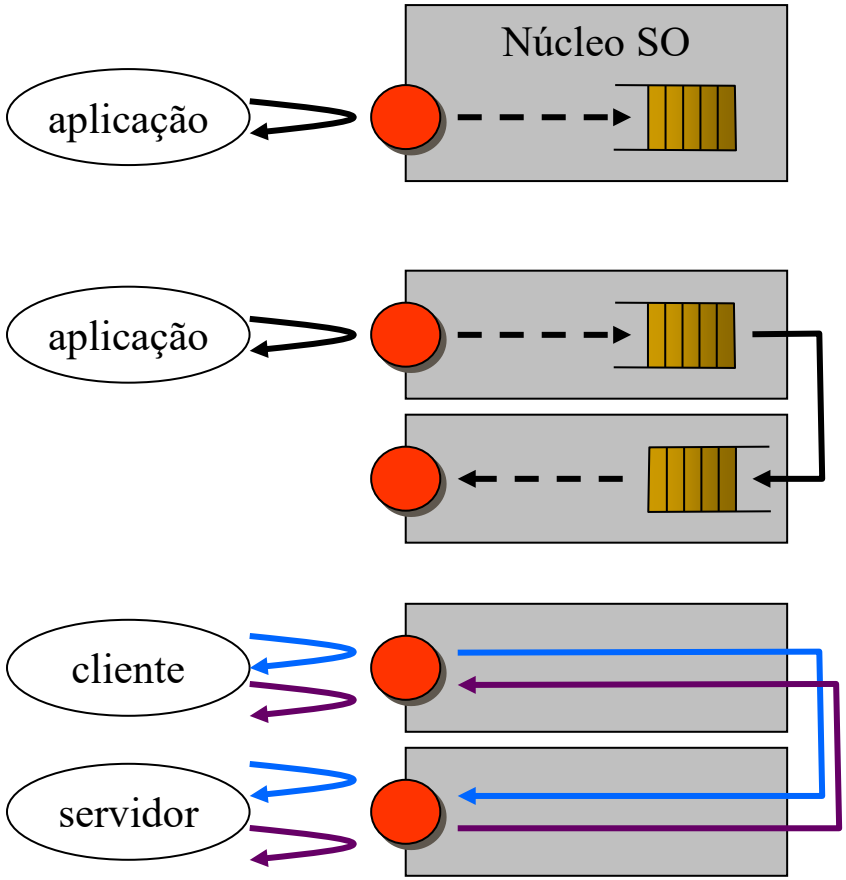
  - Normalmente não fiável

IPC: (iii) interface

- Funcionalidade normal da API dos canais
  - Com ligação: ligar/desligar, configurar, enviar/receber
  - Sem ligação: configurar, enviar/receber
- Portos
  - São extremidades de canais de comunicação
    - Em cada máquina são representados por objectos do modelo computacional local
  - Possuem 2 tipos de identificadores:
    - O do objecto do modelo computacional
      - para ser usado na API pelos processos locais
    - O do protocolo de transporte
      - para identificar a extremidade entre processos (ou máquinas) diferentes

IPC: (iv) semântica de envio

- Assíncrona
  - Transferência para os tampões do núcleo
- Síncrona
  - Garantia de entrega no destino
- Pedido/resposta
  - Cliente só prossegue após resposta do servidor



IPC: (v) semântica de recepção

- Ler de forma não bloqueante
  - Erro se nada houver na fila
- Bloquear à espera de ler uma mensagem
  - Bloqueio infinito ou temporizado
- Bloquear à espera de múltiplos eventos - guarda
  - Multiplexagem de E/S
    - e potencialmente de outras operações (ex. Windows)
  - Leitura da fila só após receber um dado evento
  - Usável para outras operações bloqueantes:
    - Espera de pedidos de ligação
    - Espera por aceitações de ligação
    - Espera por capacidade de envio

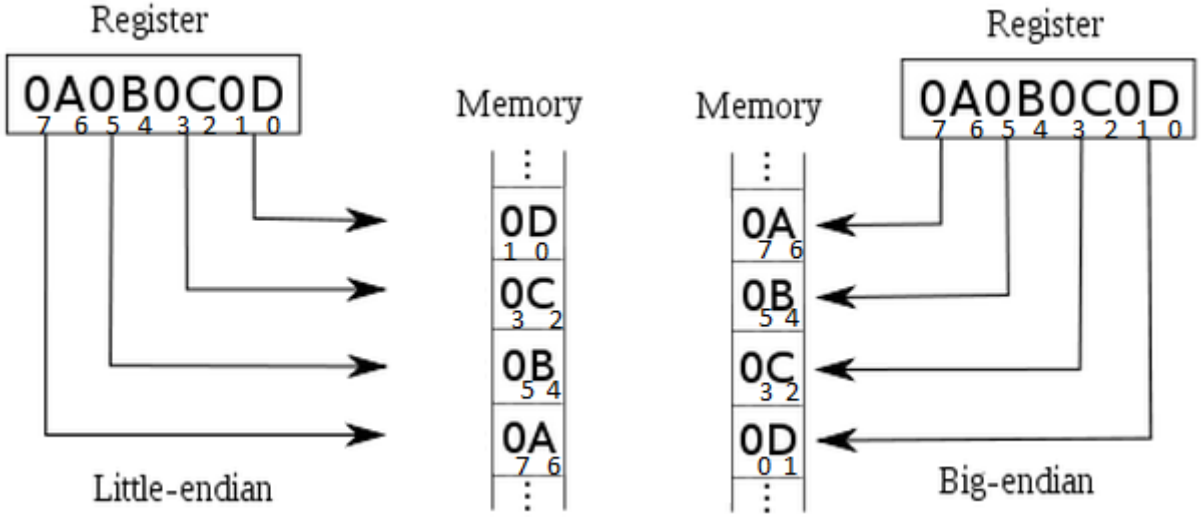
IPC: (vi) estrutura da informação

- Conteúdo
  - Estruturado ou não
  - Se for estruturado pode ser traduzido entre máquinas heterogéneas
- Fluxo
  - Sequências de bytes (*byte stream*)
    - Normalmente ilimitadas
  - Sequências de blocos de bytes
    - Fronteiras bem definidas e mantidas pelo transporte
    - Normalmente os blocos são limitados
    - O fluxo de blocos pode não manter a ordem

Representação dos Dados nas Mensagens

- A informação associada aos processos está armazenada em memória em estruturas de dados
- As mensagens enviadas são constituídas por sequências de bytes
- Quando são enviadas estruturas em mensagens, estas devem ser
  - Transformadas em sequências (*linearizadas*) antes do envio
  - Carregadas em estruturas semelhantes na memória do processo de destino
- Por outro lado, a representação interna de dados em memória depende do tipo de plataforma hardware
  - Comprimento da palavra de base (16, 32 ou 64 bits)
  - Problema do Big e Little Endian
- Além disso, há várias formas de representar caracteres
  - Sistemas de tipo Unix usam ASCII
  - A internacionalização implica a utilização de Unicode
- Torna-se pois necessário utilizar uma representação de dados independente das plataformas para permitir a transmissão dos dados
  - **Representação Normalizada de Dados**

Little-endian vs Big-endian



Exemplos:
Processadores Intel x86, RISC-V, Nios II, Andes Technology NDS32, Qualcomm Hexagon

Exemplos:
Processadores IBM Mainframe
Internet network byte order

Representação Normalizada de Dados

- Várias formas de normalizar os dados
  - Utilizar um formato comum independente da plataforma
    - Os dois intervenientes convertem os dados na emissão e recepção
  - Utilizar um formato específico indicando qual é
    - Se os dois intervenientes forem do mesmo tipo, não há conversão
- Conversão de dados
  - To *marshall*: "*to bring together and order in an appropriate or effective way*" (Webster): linearizar os dados para transmissão
  - To *unmarshall*: a operação inversa
- Exemplos de representações normalizadas
  - Sun RPC: eXternal Data Representation =► XDR
  - Corba: Common Data Representation =► CDR
  - Java RMI: Object Serialization
  - Web Services: XML

Corba Common Data Representation

- O CDR define tipos de dados básicos ou primitivos
  - octet, short, long, u_short, u_long, float, double, char, boolean, any
- Define uma representação Big e Little Endian
  - Os dados são transmitidos no formato do emissor, que é especificado em cada mensagem
  - O receptor converte se tiver uma convenção diferente
- Define também um conjunto de tipos compostos
  - Definidos a partir dos tipos básicos

Tipo	**Representação**
Sequence	Comprimento seguido dos elementos
String	Comprimento seguido dos elementos
Array	Elementos por ordem
Struct	Ordem de definição dos elem. Básicos
Enumerated	Unsigned long
Union	Tipo seguido do elemento

Exemplo Mensagem CORBA

index in sequence of bytes	← 4 bytes →	*notes on representation*
0–3	0005	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	0006	*length of string*
16–19	"Lond"	*'London'*
20-23	"on__"	
24–27	1934	*unsigned long*

The flattened form represents a *Person* struct with value: {'Smith', 'London', 1934}

- A definição da estrutura *person* é feita utilizando a linguagem de definição de interfaces do Corba (IDL)
- O código de *marshalling* é gerado automaticamente pelo compilador de interfaces

Exemplo de mensagem XML

- Definição de uma estrutura “person”

```
<person id="123456789">
                    <name>Smith</name>
                    <place>London</place>
                    <year>1984</year>
                    <!-- a comment -->
</person >
```

- Utilização de um namespace XML

```
person pers:id="123456789" xmlns:pers = "http://www.cdk5.net/person">
        <pers:name> Smith </pers:name>
        <pers:place> London </pers:place >
        <pers:year> 1984 </pers:year>
</person>
```

Exemplo de mensagem XML

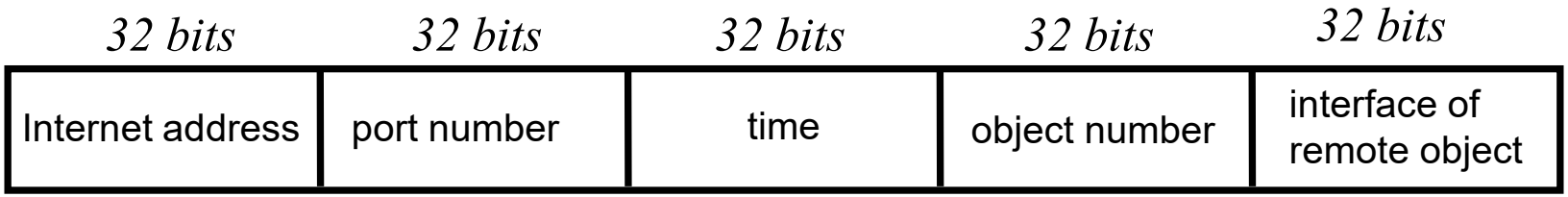
- Definição do schema XML de person:

```
<xsd:schema  xmlns:xsd = URL of XML schema definitions >
   <xsd:element name="person" type="personType" />
      <xsd:complexType name="personType">
         <xsd:sequence>
            <xsd:element name="name" type="xs:string"/>
            <xsd:element name="place" type="xs:string"/>
            <xsd:element name="year" type="xs:positiveInteger"/>
         </xsd:sequence>
         <xsd:attribute name="id" type="xs:positiveInteger"/>
      </xsd:complexType>
</xsd:schema>
```

http://www.w3schools.com/schema/default.asp

Referências de Objectos Remotos

- Quando um cliente invoca um método numa interface ou objecto remoto, tem de o designar de forma inequívoca através de um identificador
  - O identificador é igualmente passado como argumento da invocação
- Essa identificação é uma referência para o objecto remoto
  - Deve ser válida em todo o universo do sistema distribuído
  - Deve ser única no espaço e no tempo, não devendo ser reutilizada
- Exemplo de identificador:



IPC: (vii) deteção e tratamento de faltas

- O modelo de faltas depende do tipo dos canais
  - Com ligação:
    - Na ligação: destinatário inexistente, etc.
    - Na conversação: quebra de ligação, etc.
  - Sem ligação:
    - destinatário inexistente, perda de dados, etc.
- A notificação das faltas às aplicações depende da API e do modelo computacional
  - Valores de retorno da API
  - Chamadas assíncronas de procedimentos
  - Chamadas próprias da API

IPC: (viii) difusão de mensagens

- Semântica
  - Enviar apenas uma mensagens para múltiplos receptores
- Suporte à difusão depende da rede
  - Fácil em LANs (Ethernet, etc.)
    - Suporte dos níveis 1 e 2 (físico e rede)
  - Complexo em redes maiores (MANs, WANs)
    - Requer suporte dos níveis superiores
- Suporte específico a níveis superiores
  - IP *multicast*
  - Comunicação em grupo (ex. ISIS)

IPC: (ix) Exemplos de API

- UNIX
  - Sockets (BSD 82)
  - TLI (*Transport Layer Interface*, ATT 86)
  - Streams (Ritchie 84, ATT 89)
- Windows
  - NetBIOS (IBM 84)
  - NetBEUI (IBM 85)
  - Winsocks (*Windows Sockets*)
    - V1 (MS 93)
    - V2 (MS 96)
  - Named Pipes (IBM OS/2)
  - Mailslots (IBM OS/2)
  - NetDDE (MS)

IPC: (x) Exemplos de mapeamentos API → protocolos

	API	Protocolos de rede
UNIX	**Sockets**	**TCP/UDP** **IP (ICMP, etc.)** **Xerox NS**
	TLI	
	Streams	
Windows	**Winsocks**	**TCP/UDP** **NWLink → IPX/SPX**
	NetBIOS / NetBEUI	**NBT → TCP/IP** **NBF** **NWLink → IPX/SPX**
	Named Pipes	
	Mailslots	
	NetDDE	

Perguntas a que devo ser capaz de responder

- Quais são as funções dos elementos participantes na comunicação entre processos, incluindo processo, canal e porto ?
- Porque é necessário um protocolo de representação de dados ? Que ações são efetuadas no emissor e no recetor para adequar as mensagens a esse protocolo ?
- Quais são as diferenças, vantagens e desvantagens de um protocolo de representação de dados como o XML, comparando com um protocolo como o Corba ?