Computação Distribuída – Cap. III

Licenciatura em Engenharia Informática

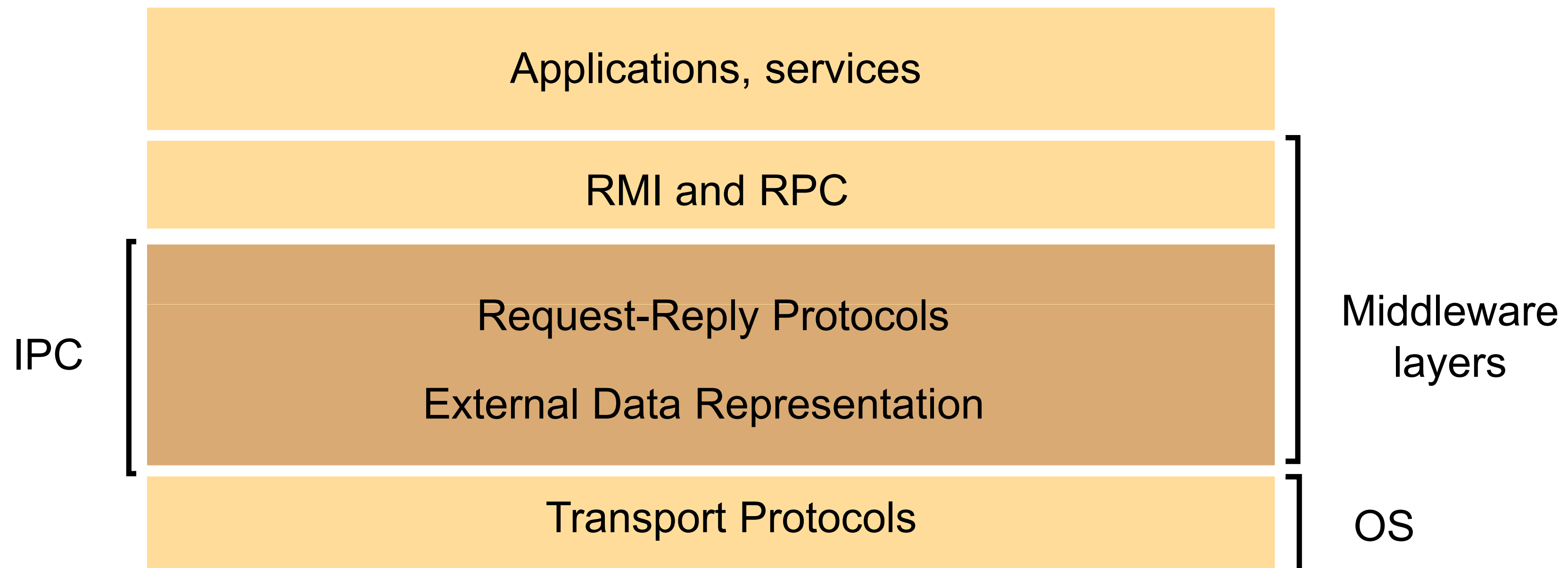
Universidade Lusófona

Prof. José Rogado
Prof. José Faísca

Comunicação entre Processos Distribuídos

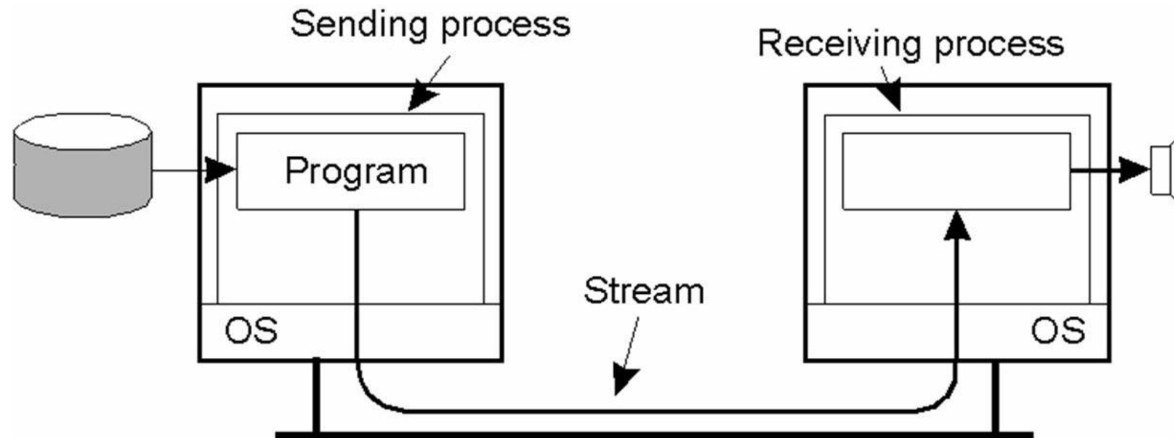
- Comunicação entre processos
  - Transporte, streams e mensagens
  - Modelos de IPC
- Comunicação
  - Ponto a ponto
  - Multi-ponto
- Invocação de métodos remotos
  - Passagem de Parâmetros
  - Heterogeneidade de Dados
- Representação Normalizada de Dados

IPC - Comunicação entre Processos



- O IPC fornece um paradigma de 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*)

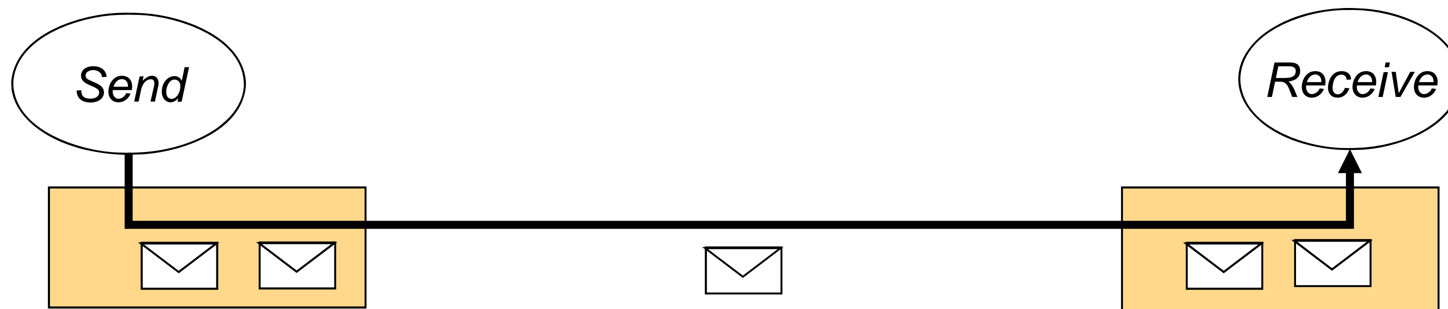
Comunicação por *Streams* (fluxos)



- Dois processos estabelecem um canal de comunicação
  - Emissor escreve de forma assíncrona num *buffer* de dados
  - Receptor pode bloquear à espera de *buffer* cheio
- Os dados são enviados de forma não estruturada
  - O canal pode ou não garantir a fiabilidade e ordenação dos pacotes de dados
    - ex: TCP vs. UDP
- Há necessidade de sincronização e ordenação entre emissor e receptor
  - Pode ser feita pelo protocolo (ex: controle de fluxo TCP)
  - Ou a nível aplicacional (ex: *stream* de áudio ou vídeo RTP ou RTSP)
- Noções de qualidade de serviço
  - Limites temporais para transmissão de dados
  - Reserva de largura de banda (ex: RSVP)

Comunicação por Mensagens

- Geralmente implementada pelas funções
  - *Send*(): envia uma sequência de bytes (estruturada ou não)
  - *Receive*(): recebe a sequência de bytes enviada
  - Exemplo: send() e receive() da API socket.
- Filas de espera de envio e de recepção de mensagens
  - O *produtor* insere mensagens da fila de emissão
  - O *receptor* retira a mensagens da fila de recepção



Comunicação Ponto-a-Ponto e Multiponto

- Normalmente a comunicação entre processos de um sistema distribuído é realizada ponto-a-ponto
  - Em emissor e um receptor
- Pode haver casos em que é vantajoso enviar simultaneamente a mesma mensagem a vários receptores
  - Por exemplo para um conjunto de servidores que implementam o mesmo serviço

    =► **Comunicação em Grupo**
  - Um grupo é um conjunto de identificadores de comunicação que é considerado pelo sistema como uma entidade única
  - A comunicação pode ser efectuada para todos (*broadcast*) ou só para um grupo (*multicast*)
  - Um grupo, assim como os seus membros, pode ser definido dinamicamente
- As razões para este tipo de comunicação são várias
  - Tolerância a falhas baseadas em servidores replicados
  - Localização rápida de serviços de Registo ou Directório
  - Actualização de cópias de dados replicados
  - Propagação de eventos de notificação

Definição de Grupos de Multicast

- Nível Ligação
  - Broadcast e Multicast Ethernet Ex: ARP envia para ff:ff:ff:ff:ff:ff
- Nível rede
  - IP Multicast: endereços classe D (224.0.0.0 a 239.255.255.255)
  - Hosts podem aderir ou deixar grupos de multicast
  - A nível aplicação só é acessível por UDP
- Nível Sistema Operativo
  - Mach e Chorus (sistemas operativos distribuídos) fornecem a noção de grupos de processos que podem receber mensagens
- Nível Middleware
  - A noção de grupo é definida pela plataforma Ex: JGroups
- Middleware de Messaging
  - Noção de Publish/Subscribe
  - JMS, IBM MQSeries, BEA MessageQ, Microsoft MSMQ

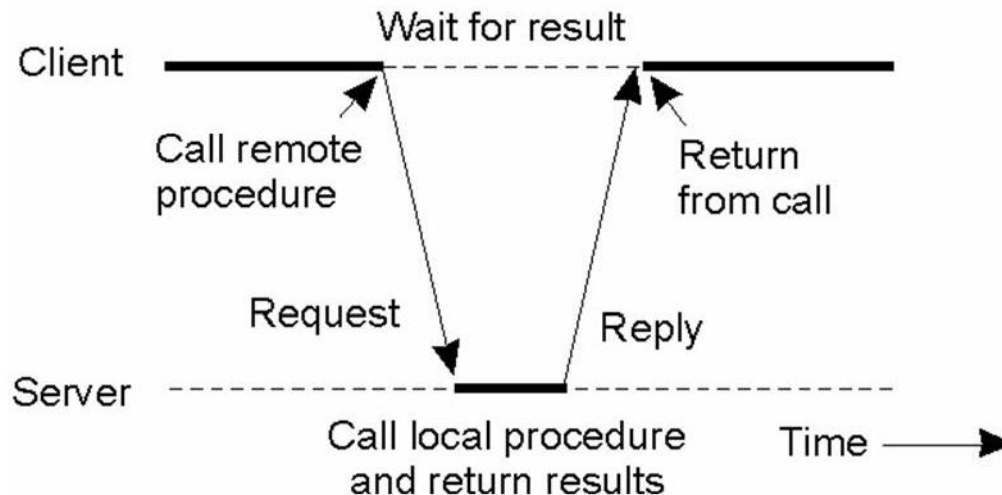
Características do Multicast

- Implementação de *Multicast*:
  - Identificador do grupo (ex.: endereço classe D)
  - Funcionalidades de criação e extinção de grupos
  - Primitivas para inserir, remover elementos do grupo
  - Possibilidade de disseminar mensagens para os elementos do grupo
    - Pode ser implementada por camadas mais baixas de protocolo
    - Ex: IP Multicast
- Fiabilidade
  - *Multicast* não fiável: não há garantia de entrega das mensagens aos elementos do grupo
    - Pode ser usado em sistemas de descoberta
  - *Multicast* fiável: entregue a todos os elementos ou a nenhum deles para evitar incoerências de estado
    - Tem de ser utilizado nos sistemas com dados e funcionalidade replicados
- Ordenação
  - Sem ordem: as mensagens podem ser entregues ao grupo em qualquer ordem
  - Ordenação total: entregues pela mesma ordem a todos os processos do grupo

Comunicação Síncrona e Assíncrona

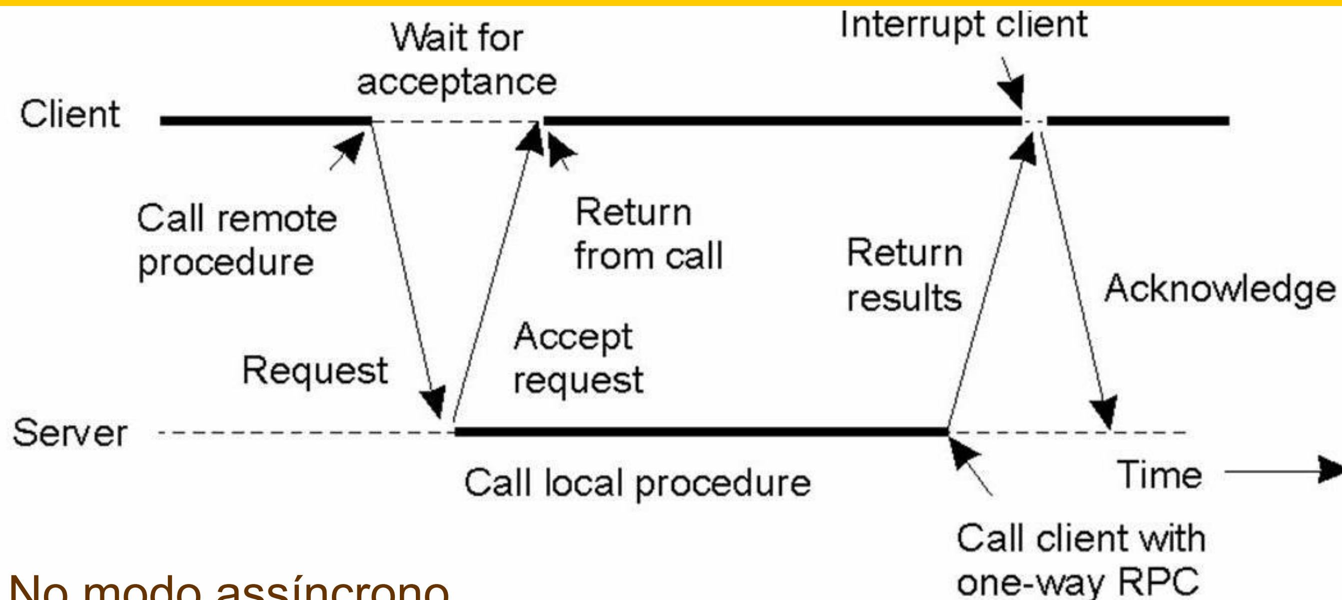
- Comunicação Síncrona
  - O *produtor* bloqueia no *send*() até a mensagem ser recebida pelo *consumidor*
  - O *consumidor* bloqueia no *receive*() até receber uma mensagem
  - A sincronização e ordenação de mensagens é garantida
  - A concorrência do sistema é limitada
- Comunicação Assíncrona
  - O produtor insere a mensagem na fila de emissão e continua
  - O consumidor é prevenido (por *evento* ou por *polling*) de que há mensagens na fila de recepção
  - A sincronização e ordenação necessita numeração das mensagens
  - A concorrência do sistema é melhorada
- A comunicação síncrona pode ser tornada assíncrona através do uso de vários fluxos de execução num processo
  - O uso de *threads* permite que uma operação bloqueante seja realizada num processo sem que este bloqueie

Pedido / Resposta Síncrono



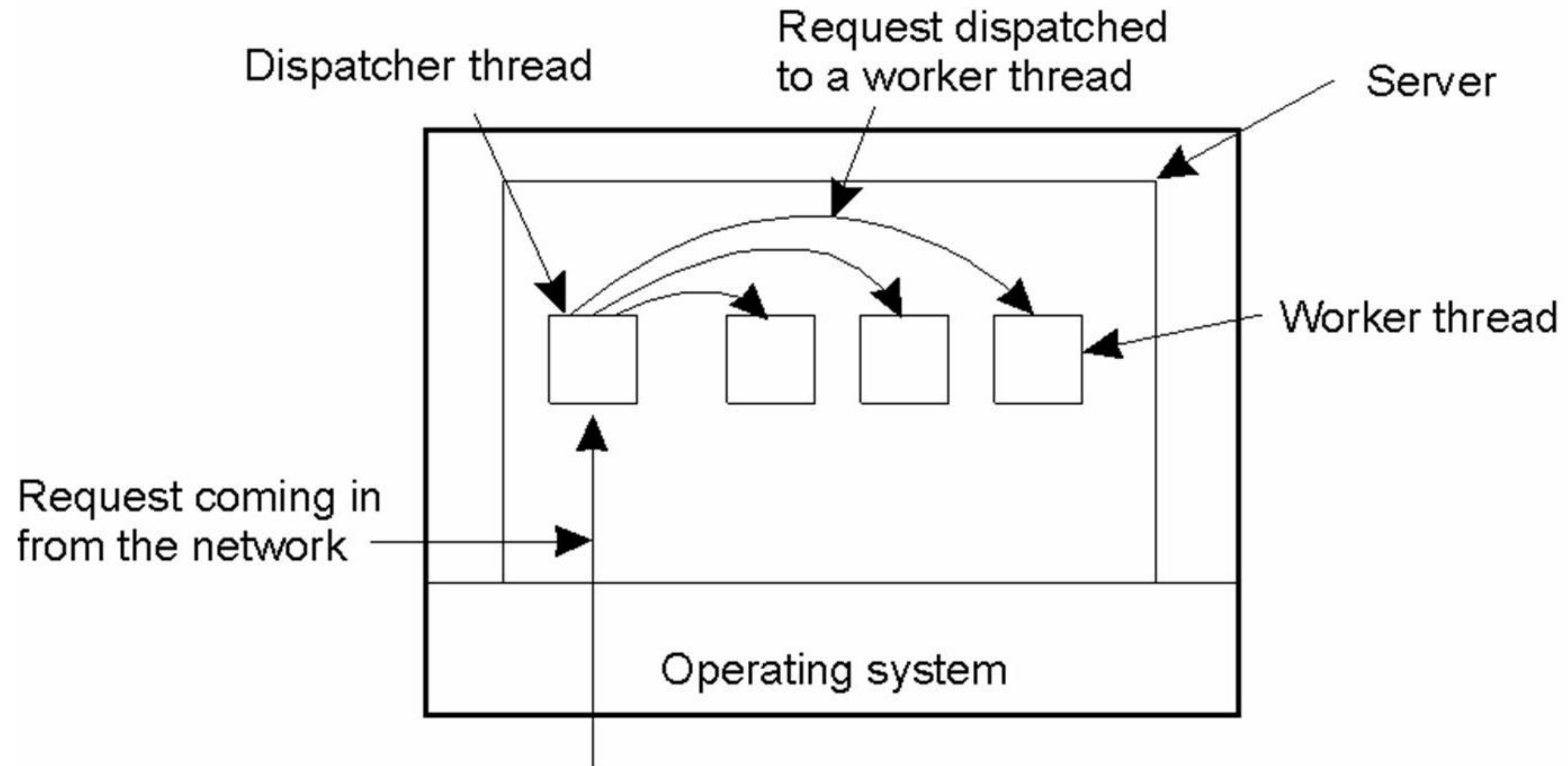
- Composta por duas mensagens distintas: *Request* e *Reply*
  - Característica do modelo RPC: *Remote Procedure Call*
- No modo síncrono
  - O *emissor* bloqueia até receber a mensagem de resposta
  - O *receptor* pode ou não bloquear até receber a mensagem, processa-a e envia a resposta
- Sincronização e Ordenação
  - Realizadas pelo fluxo de mensagens
- Concorrência
  - Bloqueio do emissor e receptor

Pedido / Resposta Assíncrono



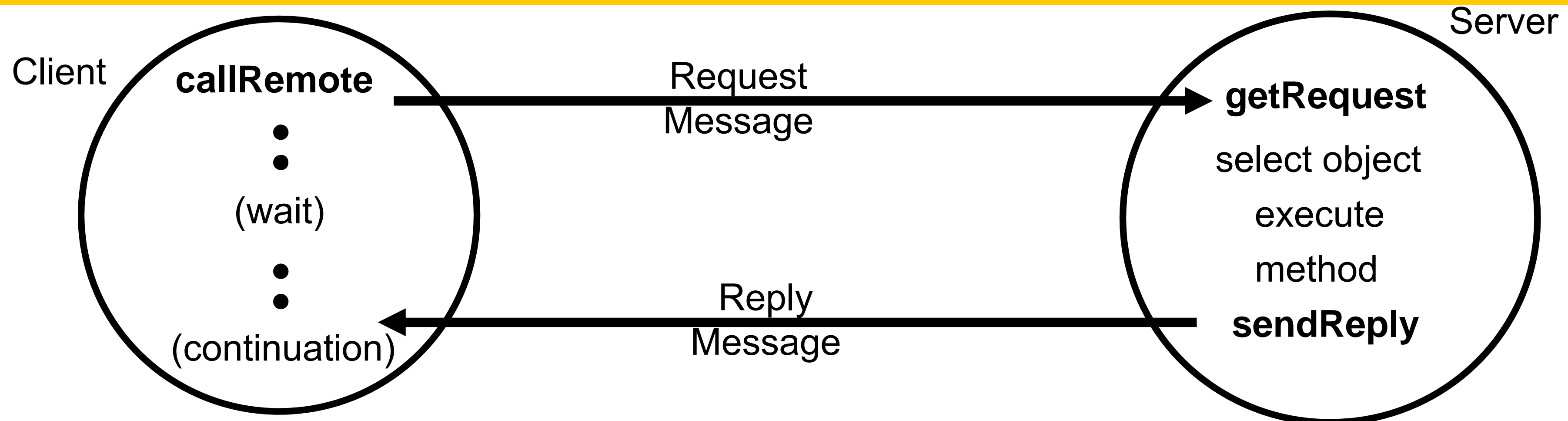
- No modo assíncrono
  - O *emissor* bloqueia só até obter garantia de entrega da mensagem
  - O *receptor* pode ou não bloquear até receber a mensagem
  - O *receptor* processa a mensagem e envia a resposta numa mensagem específica de retorno
- Característica de modelos de IPC avançados
  - ex: sistema Mach 3.0
- Sincronização e Ordenação
  - Realizadas pelas aplicações
  - Pode ser obtida a partir do modelo síncrono utilizando *threads* dedicadas

Serviço IPC assíncrono multi-threaded



- A actividade de recepção é desacoplada do tratamento dos pedidos
- Existe uma estrutura semelhante do lado do receptor

Análise do Protocolo Pedido / Resposta



- É baseado em 3 métodos genéricos:
  - callRemote: é utilizado pelo cliente para invocar uma operação remota
    - Os seus argumentos incluem um identificador do objecto remoto, o método a invocar e os respectivos parâmetros de chamada
    - Envia os argumentos numa mensagem e fica à espera da resposta
  - getRequest: é utilizado pelo servidor para receber pedidos
    - Recebe a mensagem, extrai o identificador do objecto e os respectivos argumentos e invoca o método indicado
    - Quando o método retorna, recupera os argumentos e envia-os ao emissor através de sendReply
  - sendReply: é utilizado pelo servidor para enviar a resposta ao cliente

Semântica de Invocação

- A semântica de uma invocação indica a forma como são executados os vários passos que a compõem e quais as possíveis acções e consequências em caso de falha
- Falhas por omissão e ordenação
  - Se o protocolo de transporte não garante entrega ordenada das mensagens o middleware de invocação deve gerir retransmissões e números de sequência
- Falhas Temporais
  - A ocorrência de *timeouts* pode significar perca de mensagem ou *crash* do serviço
- Em caso de falha, uma invocação pode ser realizada múltiplas vezes
  - O efeito da invocação repetida tem de ser gerido correctamente de forma a evitar incoerências no resultado
- O tipo da operação condiciona a semântica da invocação
  - Operações **idempotentes** podem ser realizadas várias vezes
    - Ex. leitura de valores, i.e.; consulta de saldo
  - Operações **não idempotentes** só podem ser realizadas uma vez
    - Ex: escrita e acumulação de valores, i.e.; depósito numa conta

Semântica de Invocação (ii)

- A sequência *callRemote* - *sendReply* pode ser implementada de várias formas:
  - Reenvio da mensagem até recepção da resposta ou assumir que o serviço falhou
  - Filtragem de mensagens duplicadas no servidor
  - Possibilidade de retransmissão de resultados mantendo um historial das invocações
- A combinação destas opções pode levar a várias possibilidades da semântica da garantia de realização da invocação remota

Medidas de Tolerância a Falhas			Semântica de Invocação
Retransmissão do pedido	**Filtragem de Duplicados**	**Acção de resposta**	
Não	**N/A**	**N/A**	**Talvez**
Sim	**Não**	**Reexecução do método**	**Pelo menos uma vez**
Sim	**Sim**	**Retransmissão da resposta**	**No máximo uma vez**

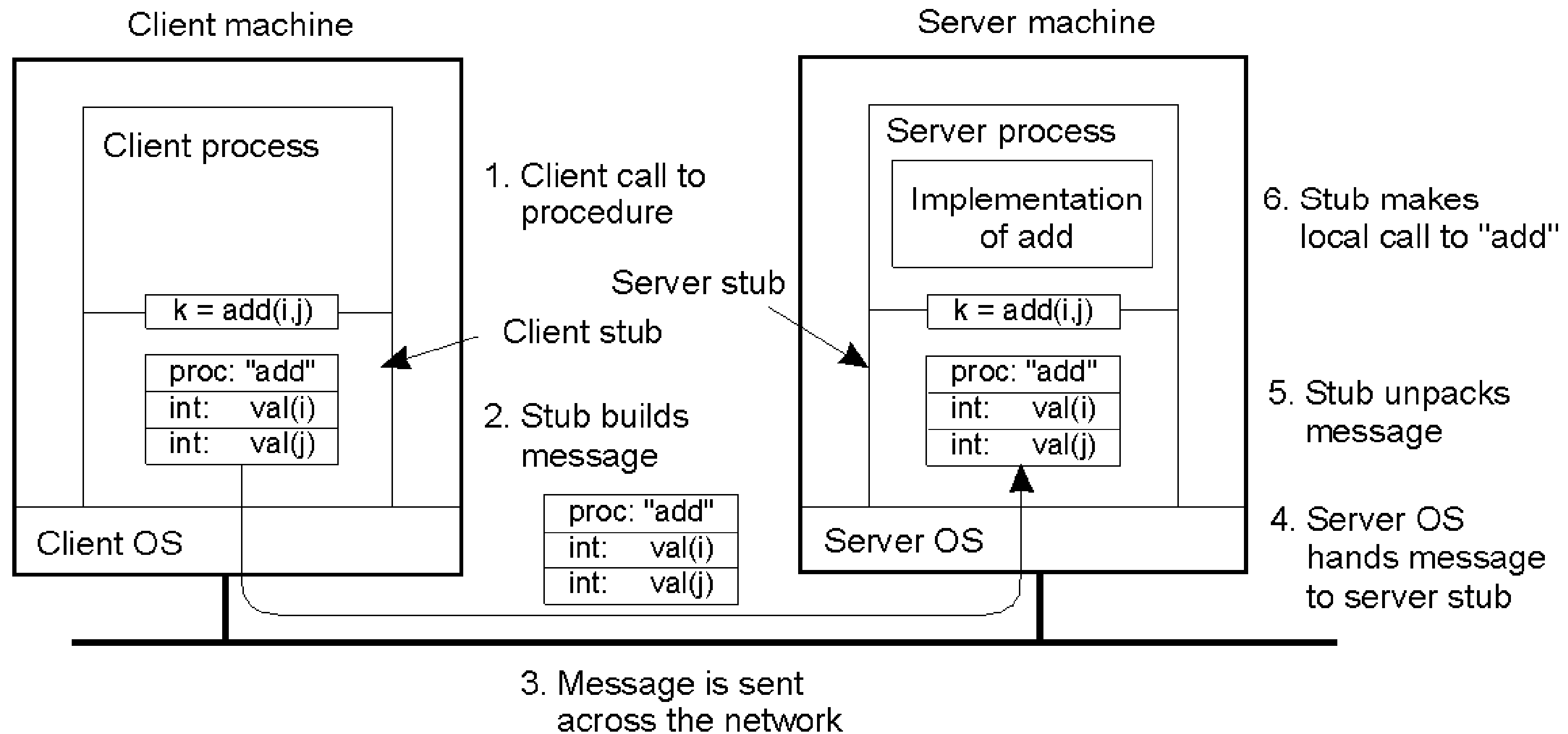
Semântica de Invocação (iii)

- *Maybe*
  - O método remoto pode ou não ser invocado, dependendo de existirem ou não falhas no sistema
    - Falhas de omissão no canal
    - Fail-stop do processos servidor
  - É o grau zero de tolerância a falhas, só aceitável em sistemas específicos com muita redundância (Ex. DNS)
- *At-least-once*
  - Receber a resposta indica que o método remoto foi executado, mas não se sabe exactamente quantas vezes
  - Receber uma excepção indica que o método não foi executado de todo
    - Falhas arbitrárias no canal podendo ter levado a retransmissão da mensagem provocando várias execuções da mesma acção -> resultados incoerentes
  - O RPC Sun tinha este tipo de semântica inicialmente
- *At-most-once*
  - Receber a resposta indica que o método foi executado uma só vez
    - Os reenvios duplicados são rejeitados pelo servidor, que reenvia o resultado
  - Receber uma excepção indica que o método não foi executado de todo
  - Java RMI e Corba fornecem este tipo de semântica
  - Versão actuais do SUN RPC

Gestão da Repetição

- Caso haja retransmissão devido a atraso no canal, uma invocação pode ser recebida várias vezes pelo serviço
  - Se a repetição não for gerida pode causar incoerência
- A gestão da repetição pode ser realizada de várias formas
  - Utilizando um contador de invocações para evitar a repetição de uma acção
  - Garantindo que as invocações podem ser realizadas várias vezes sem introduzir incoerência
- Semântica *at-least-once*
  - Neste caso a garantia dada é que as invocações são executadas pelo menos uma vez : as invocações ***têm de ser*** idempotentes
- Semântica *at-most-once*
  - Neste caso a garantia é de que as invocações só são executadas exactamente uma vez : as invocações ***podem não ser*** idempotentes

Sequência de Invocação de um RPC



Estrutura de uma Mensagem

Campo	Utilização
msgType	Tipo da mensagem Ex: request = 0; reply= 1
requestID	Identificador do pedido
objectReference	Referência do objecto remoto
methodID	Identificador do método invocado
Argumentos do Método	Sequência de bytes em formato independente*

* Heterogeneidade

- Para que possa haver invocação entre diferentes plataformas, os argumentos do método têm de ser representados num formato independente

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**

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 XDR

- Definição da estrutura *person*
- Utilização do utilitário *rpcgen*
- Geração automática do código de *marshalling*

```
typedef string nametype<MAXNAMELEN>;

struct person {
     nametype name;
     nametype place;
     int year;
};
```

```
#include "person.h"

bool_t
xdr_nametype (XDR *xdrs, nametype *objp)
{
    register int32_t *buf;

     if (!xdr_string (xdrs, objp, MAXNAMELEN))
         return FALSE;
    return TRUE;
}

bool_t
xdr_person (XDR *xdrs, person *objp)
{
    register int32_t *buf;

     if (!xdr_nametype (xdrs, &objp->name))
         return FALSE;
     if (!xdr_nametype (xdrs, &objp->place))
         return FALSE;
     if (!xdr_int (xdrs, &objp->year))
         return FALSE;
    return TRUE;
}
```

Utilização de XML: Extensible Markup Language

- Linguagem de Markup definida pelo W3C
  - Markup: representação de texto e propriedades associadas
  - Define estrutura ou representação gráfica
- Os dados em XML têm descritores associados que indicam a sua estrutura
  - O conjunto de descritores é extensível
  - Utilização de Namespaces para definir conjuntos específicos de descritores
- É utilizado universalmente para definir estruturas de dados e interfaces
- Na Web o XML é usado para definir interfaces e dados dos Web Services
  - EX: SOAP
- A estrutura dos elementos e atributos dos documentos pode ser definida utilizando DTDs (Document type Definitions) e XML Schemas
- Exemplo
  - Estrutura person:
    - Person, name, place e year são elementos
    - Id é um atributo que é inicializado com um valor

```
<person id="123456789">
        <name>Smith</name>
        <place>London</place>
        <year>1934</year>
        <!-- a comment -->
</person >
```

XML Namespace

- A utilização de *namespaces* permite referenciar listas de elementos e atributos já existentes
- Inserção do URL do ficheiro de definição no interior do elemento que faz referencia ao *namespace*

```
<person pers:id="123456789" xmlns:pers = URL of definition>
    <pers:name> Smith </pers:name>
    <pers:place> London </pers:place >
    <pers:year> 1934 </pers:year>
</person>
```

XML Document Type Definition (DTD)

- Um DTD define a estrutura dos blocos que podem existir num documento XML
- Tem tendência a ser substituído pela noção de Schema que é mais genérica

```
<!DOCTYPE person [
    <!ELEMENT person (name, place, year)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT place (#PCDATA)>
    <!ELEMENT year (#PCDATA)>
    <!ATTLIST person id CDATA “0”>
]>
```

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

XML Schema

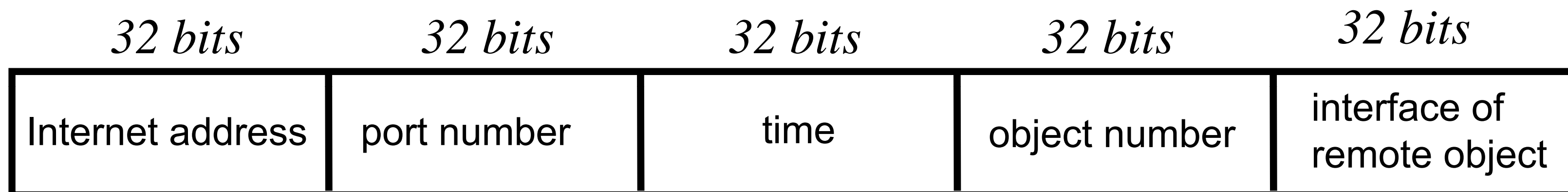
- Um schema define os elementos e atributos utilizados no documento XML. Permite:
  - Definir a ordem, estrutura e formato dos elementos
  - Definir os tipos básicos utilizados (string, integer, etc…)

```
<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:



Fim do Capítulo 3

- Resumo dos conhecimentos adquiridos
  - Comunicação entre processos
    - Fluxos
    - Mensagens
  - Tipos de Comunicação
    - Ponto a ponto
      - Síncrona e Assíncrona
      - Pedido / resposta
    - Multi-ponto
  - Invocação de métodos remotos
    - Estrutura das Mensagens
    - Passagem de Parâmetros
    - Heterogeneidade de Dados
  - Representações normalizadas de dados
    - XDR, Corba, XML
- Trabalho Individual Complementar
  - Big e Little Endian: origens, características e plataformas
    - http://en.wikipedia.org/wiki/Endianness
  - Compreender XML DTD e XML Schema
    - http://www.w3schools.com/dtd/default.asp
    - http://www.w3schools.com/schema/default.asp