Comunicação entre Processos

Ricardo Costa – rcosta@estg.ipp.pt

Comunicação entre Processos

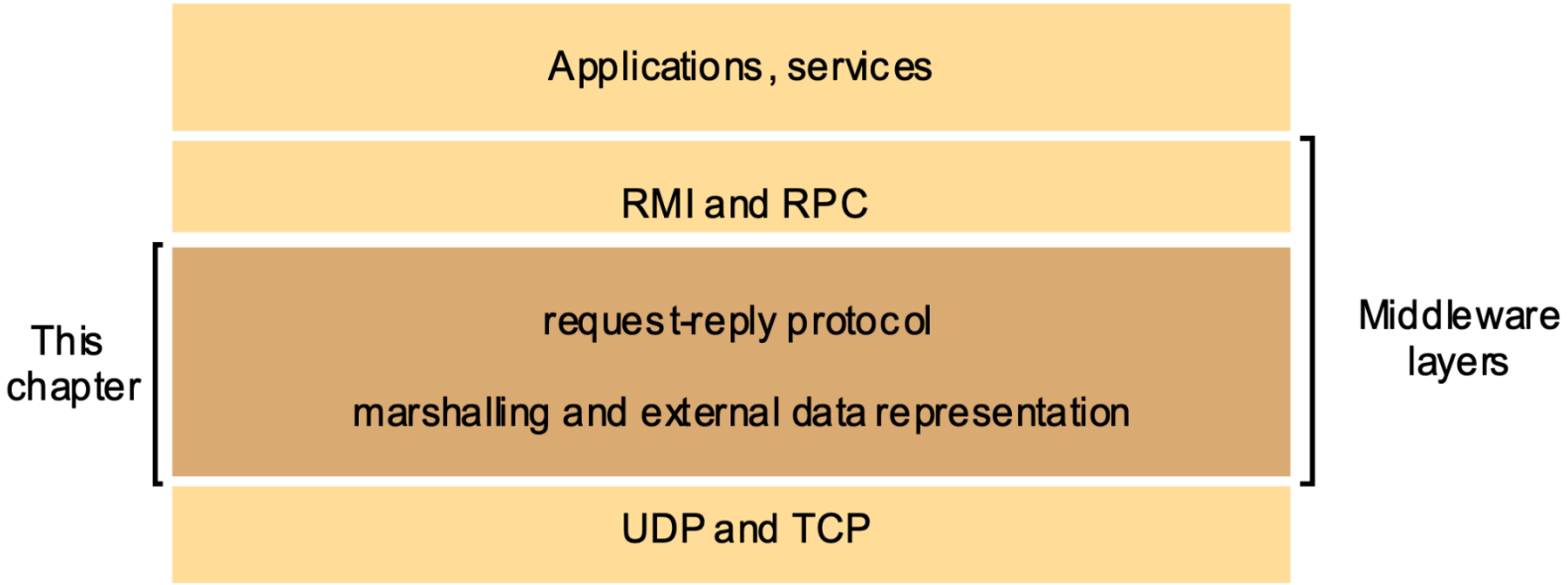
Sockets UDP e TCP

A serialização de estruturas de dados

Comunicação cliente-servidor

- Protocolo pedido-resposta
- Semântica perante falhas

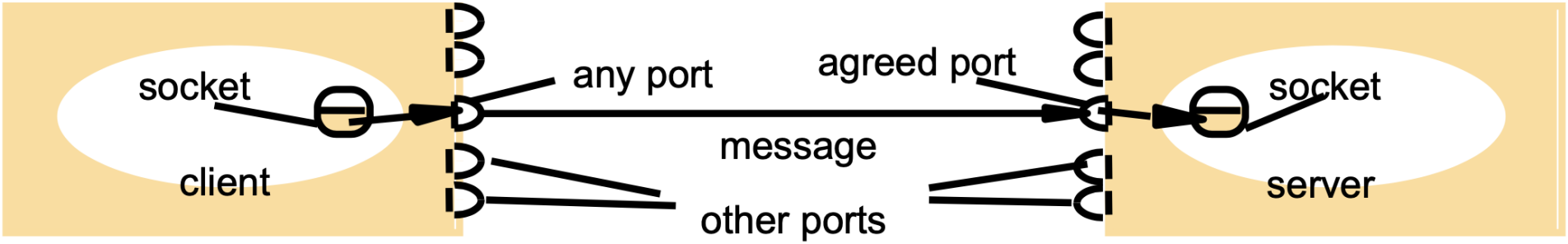
Comunicação entre Processos



Comunicação entre Processos

Sockets UDP e TCP

- Ideia surgida com o sistema UNIX de Berkeley - BSD Unix
- Abstracção para representar a comunicação entre processos:
  - A comunicação entre dois processos consiste na transmissão de uma mensagem de um socket num processo para um socket noutro processo.



Comunicação entre Processos

Sockets UDP e TCP

- Nos protocolos internet, as mensagens são enviadas para um par:
  - Endereço internet
  - Nº de um porto
- O socket de um processo tem que ser conectado a um porto local para que possa começar a receber mensagens.
- Um vez criado tanto serve para receber como para enviar mensagens.
- O número de portos disponíveis por computador é 2^{16} (= 65536)
- Para receber mensagens, um processo pode usar vários portos simultaneamente, mas não pode partilhar um porto com outro processo diferente no mesmo computador.
  - Excepção: processos que usem IP multicast
- Cada socket é associado a um determinado protocolo, UDP ou TCP.

Comunicação entre Processos

```
public Cliente(){
 try {
   Socket sc = new Socket("127.0.0.1", 2222);
   System.out.println("Cliente cria socket    " + sc );
   …
Cliente cria socket    Socket[addr=/127.0.0.1,port=2222,localport=1533]

…

public Servidor() {
 try {
   ServerSocket ss = new ServerSocket(2222);
   while (true) {
     Socket s = ss.accept();
     System.out.println("Servidor - accept executado   " + s);
     System.out.println(s.getOutputStream());
     …
Processo servidor - accept executado  Socket[addr=/127.0.0.1,port=1533,localport=2222]
java.net.SocketOutputStream@45a877
```

Comunicação entre Processos

Sockets UDP e TCP

- Principais protocolos de rede actuais:
  - UDP – User Datagram Protocol
    - protocolo sem conexão
  - comunicação por “datagrams”
  - TCP – Transmission Control Protocol
    - protocolo com conexão
    - comunicação por streams

Comunicação entre Processos

Sockets UDP e TCP

- Comunicação através do protocolo UDP
  - A comunicação entre dois processos é feita através dos métodos send e receive.
  - Um item de dados ("datagram") é enviado por UDP sem confirmação ("acknowledgment") nem reenvio.
  - Qualquer processo que queira enviar ou receber uma mensagem tem que criar um socket com o IP da máquina local e o número de um porto local.
  - O porto do servidor terá que ser conhecido pelos processos clientes.
  - O cliente pode usar qualquer porto local para conectar o seu socket.
  - O processo que invocar o método receive (cliente ou servidor) recebe o IP e o porto do processo que enviou a mensagem, juntamente com os dados da mensagem.
  - Tamanho da mensagem:
    - O receptor da mensagem, tem que definir um array (buffer) com dimensão suficiente para os dados da mensagem
    - O protocolo permite mensagens até 216 bytes.
    - Mensagens maiores que o buffer definido, serão truncadas

Comunicação entre Processos

Sockets UDP e TCP

- Comunicação através do protocolo UDP
  - Operações não bloqueáveis:
    - send
      - O processo retorna do send assim que a mensagem é enviada. No destino, a mensagem é colocada na fila do socket respetivo.
      - Se nenhum processo estiver ligado ao socket, a mensagem é descartada.
  - Operações bloqueáveis:
    - receive
      - O processo que executa o receive, bloqueia até que consiga ler a mensagem para o buffer do processo.
      - Enquanto espera por uma mensagem o processo pode criar uma nova thread para executar outras tarefas.
      - Ao socket pode ser associado um timeout, findo o qual o receive desbloqueia.

Comunicação entre Processos

Sockets UDP e TCP

- Comunicação através do protocolo UDP
  - Modelo de Avarias
    - Tipo de avarias que podem ocorrer:
      - Avaria por omissão – a mensagem não chega porque,
      - Buffer cheio local ou remotamente
      - Erro de conteúdo - checksum error
      - Avaria de ordenamento – as mensagens chegam fora de ordem

Comunicação entre Processos

Sockets UDP e TCP

- Utilização do protocolo UDP:
  - Aplicações onde são aceitáveis avarias de omissão
  - Domain Naming Service - DNS
  - Transmissão de imagem
  - ...
- Classe DatagramSocket em Java
  - permite criar um socket na máquina local para o processo corrente
  - construtor sem argumentos, usa o primeiro porto disponível
  - construtor com argumentos especifica-se o nº do porto
  - se o porto está a ser usado é gerada a excepção SocketException
  - ...

Comunicação entre Processos

Sockets UDP e TCP

- Classe DatagramPacket em Java
  - Ao instanciar um DatagramPacket para enviar uma mensagem, usar o construtor com os parâmetros:
    - Um array de bytes que contém a mensagem,
    - O comprimento da mensagem
    - O endereço Internet do socket destino
      - Objecto do tipo InetAddress
    - Nº do porto do socket destino
  - Ao instanciar um DatagramPacket para receber uma mensagem, usar o construtor com os parâmetros:
    - Referência de um buffer de memória para onde a mensagem será transferida,
    - O comprimento desse buffer
    - A mensagem é colocada neste objecto do tipo DatagramPacket.
    - Para extrair os dados da mensagem usa-se o método
      - getData() da classe DatagramPacket
      - Os métodos getPort() e getAddress() devolvem o nº do porto e o IP do processo emissor, respectivamente.

Comunicação entre Processos

Sockets UDP e TCP

- Exemplo de utilização de sockets UDP em Java:
  - Um processo cliente envia uma mensagem para um nó remoto e recebe em resposta a mesma mensagem. A mensagem e o nome da máquina remota são passados como parâmetros do programa.
  - O processo servidor fica à espera de mensagens no porto 6789. Ao receber uma mensagem, extrai a mensagem e envia-a de volta para o cliente, para o IP e o porto recebidos.

Comunicação entre Processos

Sockets UDP e TCP

- Exemplo de utilização de sockets UDP em Java:

Processo Cliente

```
import java.net.*;
import java.io.*;
public class UDPClient{

   public static void main(String args[]){
   // args vai conter o conteúdo da mensagem e o nome do servidor
            DatagramSocket aSocket = null;
            try {
                // cria um socket para o processo cliente ligando-o a um porto disponível
                aSocket = new DatagramSocket();
                byte [] m = args[0].getBytes();
                InetAddress aHost = InetAddress.getByName(args[1]);
                int serverPort = 6789;

                //criar o datagrama para envio
                 DatagramPacket request =
                        new DatagramPacket(m, args[0].length(), aHost, serverPort);
                    // envia a mensagem
                    aSocket.send(request);
```

Comunicação entre Processos

Sockets UDP e TCP

- Exemplo de utilização de sockets UDP em Java:

Processo Cliente (cont ...)

```
// prepara o cliente para receber resposta do servidor
byte[] buffer = new   byte[1000];
DatagramPacket reply = new DatagramPacket(buffer, buffer.length);

// recebe resposta
aSocket.receive(reply);

System.out.println("Reply: " + reply.getData());


 }catch (SocketException e)
    {System.out.println("Socket: " + e.getMessage());
}catch (IOException e)
    {System.out.println("IO: " + e.getMessage());}

}finally {if(aSocket != null) aSocket.close();}

} }
```

Comunicação entre Processos

Sockets UDP e TCP

- Exemplo de utilização de sockets UDP em Java:

Processo Servidor

```
import java.net.*;
import java.io.*;
public class UDPServer{
        public static void main(String args[]){
        DatagramSocket aSocket = null;
          try{

            //cria um objecto do tipo socket e liga-o ao porto 6789
                aSocket = new DatagramSocket(6789);

                // buffer de recepção vazio
                byte[] buffer = new byte[1000];

                while(true){
                  // instancia o objecto onde vai receber a msg
                  DatagramPacket request =
                                new DatagramPacket(buffer, buffer.length);

                  // bloqueia até receber a mensagem
                  aSocket.receive(request);
```

Comunicação entre Processos

Sockets UDP e TCP

- Exemplo de utilização de sockets UDP em Java:

Processo Servidor (cont ...)

```
            // instancia o objecto para enviar a mensagem
            DatagramPacket reply =
                    new DatagramPacket(request.getData(), request.getLength(),
                                       request.getAddress(), request.getPort());

            // envia a resposta ao cliente
                aSocket.send(reply);

            } // fim do while

        }catch (SocketException e)
                {System.out.println("Socket: " + e.getMessage());
        }catch (IOException e)
                {System.out.println("IO: " + e.getMessage());}
        }finally
        {if(aSocket != null) aSocket.close();}
} }
```

Comunicação entre Processos

Sockets UDP e TCP

- Comunicação através do protocolo TCP
  - Utilização da abstracção stream para ler/escrever dados.
  - Tamanho das mensagens:
    - A aplicação é que decide quantos bytes devem ser enviados ou lidos da stream, sem a preocupação do tamanho máximo de pacotes
  - Perda de Mensagens:
    - O protocolo TCP usa um esquema de confirmação de recepção das mensagens. Se necessário retransmite a mensagem.
  - Controlo do fluxo de execução:
    - O TCP tenta uniformizar as velocidades dos processos que lêem e escrevem de/numa stream.
    - Se "quem" escreve é muito mais rápido do que "quem" lê, então o processo que escreve é bloqueado até que o outro processo leia dados suficientes

Comunicação entre Processos

Sockets UDP e TCP

- Comunicação através do protocolo TCP
  - Ordenação e duplicação de mensagens:
    - Identificadores de mensagens são associados com cada pacote de dados, permitindo ao receptor detectar e rejeitar mensagens duplicadas ou reordenar mensagens fora de ordem.
  - Destino das mensagens:
    - Um par de processos estabelece uma conexão antes de poderem comunicar por uma stream. A partir dessa ligação, podem comunicar sem terem de indicar o endereço IP nem o nº de porto.
  - Modelo de comunicação:
    - Quando dois processos tentam estabelecer uma ligação através de Sockets TCP, um dos processos desempenha o papel de cliente e outro de servidor. Depois de estabelecida a ligação podem comportar-se como processos pares.

Comunicação entre Processos

Sockets UDP e TCP

- Comunicação através do protocolo TCP
  - Cliente:
    - Cria um objecto do tipo Socket que tenta estabelecer uma ligação com um porto de um servidor, numa máquina remota. Para estabelecer esta ligação é necessário indicar o endereço IP e o porto da máquina remota.
  - Servidor:
    - Cria um objecto do tipo "listening" socket associado ao porto servidor. Este socket possui um método que fica bloqueado até que receba um pedido de ligação ao porto correspondente.
    - Quando chega o pedido de ligação, o servidor aceita-a instanciando um novo socket que, tal como o socket do cliente, tem duas streams associadas, uma para saída outra para entrada de dados.

Comunicação entre Processos

Sockets UDP e TCP

- Comunicação através do protocolo TCP
  - Modelo de avarias
    - Streams TCP usam
      - checksums para detectar e rejeitar pacotes corrompidos;
      - timeouts e retransmissão para lidar com pacotes perdidos;
      - número de sequência para detectar e rejeitar pacotes duplicados;
    - Se uma mensagem não chega porque o sistema está congestionado, o sistema não recebe a confirmação da recepção da mensagem, reenvia sucessivamente a mensagem até que a conexão é cancelada após um certo tempo.
    - A mensagem não é transmitida, os processos participantes ficam sem saber o que aconteceu:
      - Falha na rede
      - Falha do outro processo

Comunicação entre Processos

Sockets UDP e TCP

- Utilização do protocolo TCP
  - Os serviços HTTP, FTP, Telnet, SMTP, ...

Comunicação entre Processos

Sockets UDP e TCP

- A serialização de estruturas de dados
  - Tanto o processo local como o processo remoto manipulam estruturas de dados locais.
  - Para a transmissão de dados numa mensagem, é necessário serializar esses dados em sequências de bytes.
  - Do outro lado, os dados têm que ser reestruturados de forma a representarem a informação original mesmo que a arquitectura da máquina do processo receptor seja diferente da arquitectura do emissor.
  - Exemplos de diferença de formatos consoante a arquitectura
  - Valores inteiros podem ser representados com:
    - o bit mais significativo em primeiro lugar, i.é, endereço mais baixo, (big-endian) – mainframe IBM; ou
    - o bit mais significativo no fim (little-endian) – processadores intel (i. é, o bit menos significativo no endereço mais baixo)
    - Valores reais, formato IEEE574 – processadores intel formato BDC – processador mainframe da IBM
    - Valores carácter, um char, 1 byte – Unix, um char, 2 bytes - Unicode
  - A heterogeneidade do hardware obriga à utilização de formatos neutros de serialização.

Comunicação entre Processos

Sockets UDP e TCP

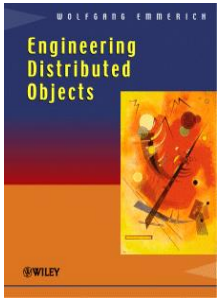
- A serialização de estruturas de dados
  - Duas formas de permitir que quaisquer computadores diferentes troquem valores:
    - Ter uma representação externa comum para os dois.
      - Os valores são convertidos para a representação externa, e depois, no receptor, são convertidos para o formato do receptor.
      - Se os dois computadores são iguais poderá omitir-se a conversão.
    - Não ter a representação externa, mas junto com os dados é enviada informação sobre o formato usado, de forma a que o receptor possa converter os valores se necessário.

Comunicação entre Processos

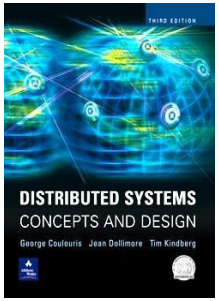
Sockets UDP e TCP

- A serialização de estruturas de dados
  - Duas formas de permitir que quaisquer computadores diferentes troquem valores:
    - Ter uma representação externa comum para os dois.
      - Os valores são convertidos para a representação externa, e depois, no receptor, são convertidos para o formato do receptor.
      - Se os dois computadores são iguais poderá omitir-se a conversão.
    - Não ter a representação externa, mas junto com os dados é enviada informação sobre o formato usado, de forma a que o receptor possa converter os valores se necessário.

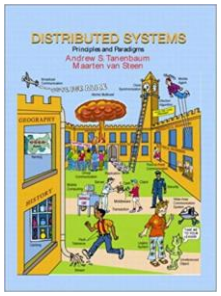
Bibliografia



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?