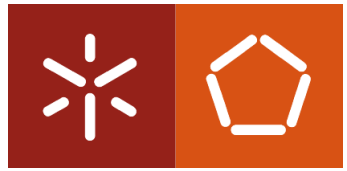
Programação com Sockets



Objectivo: construir aplicações cliente/servidor que usam sockets

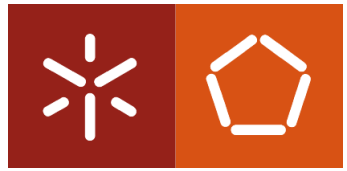
Socket API

- Introduzida no UNIX BSD4.1 em 1981
- Os sockets devem ser explicitamente criados usados e libertados pelas aplicações
- Paradigma cliente/servidor
- Dois tipos de serviço de transporte:
  - Datagrama não fiável
  - Stream fiável

socket

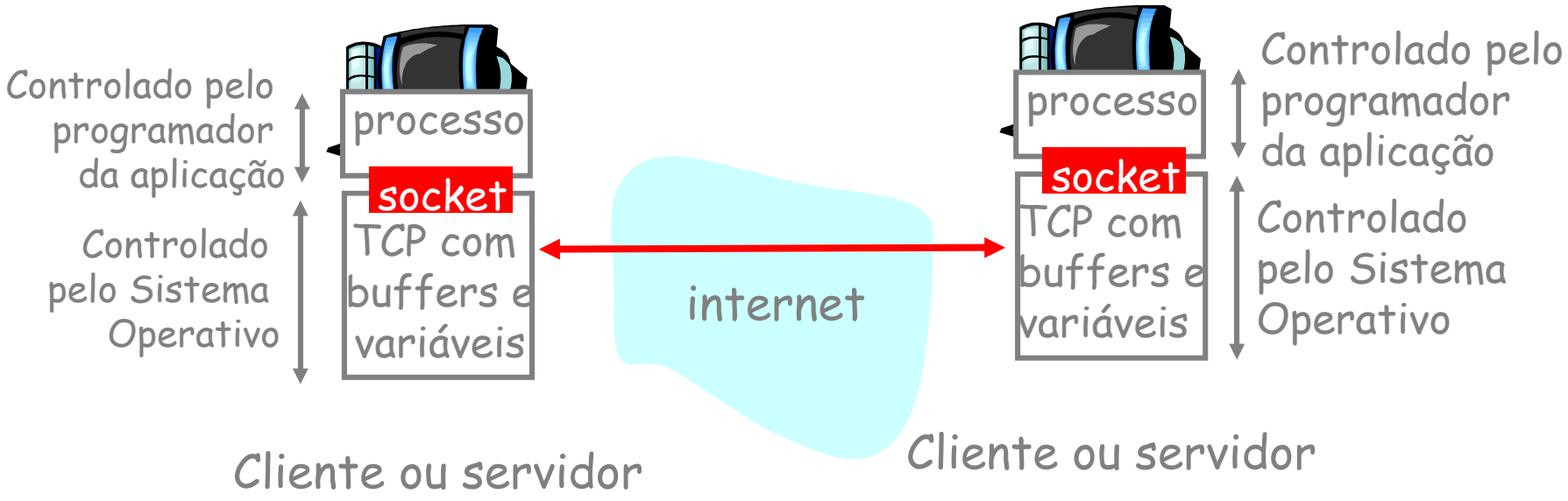
Uma interface ***local a um sistema, criado pelas aplicações, controlado pelo sistema operativo*** (uma "porta") na qual um processo de aplicação pode simultaneamente **enviar e receber** mensagens de outros processos

Programação com Sockets: TCP

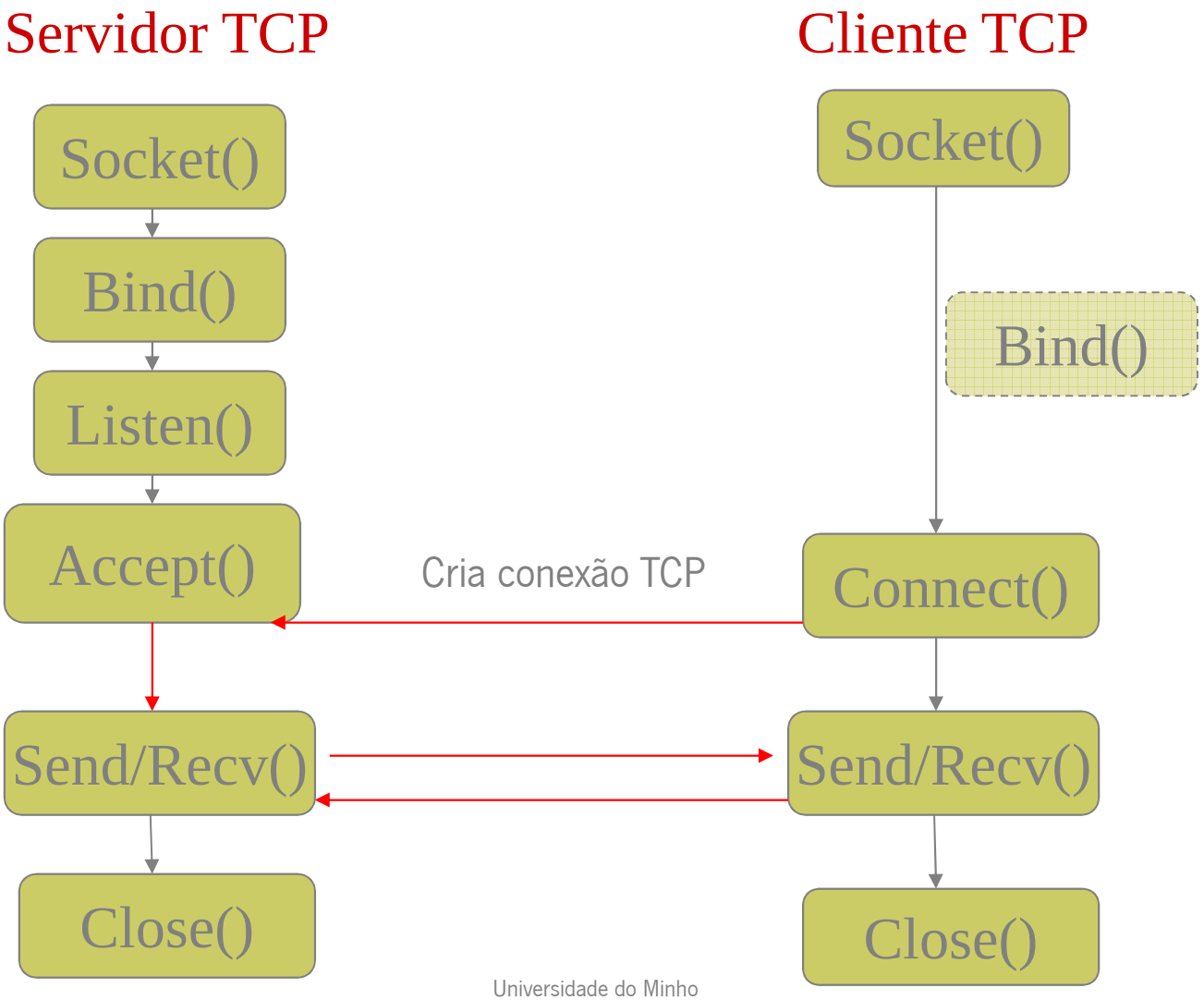


Socket: uma porta entre o processo que executa a aplicação e o protocolo de transporte fim-a-fim (UCP or TCP)

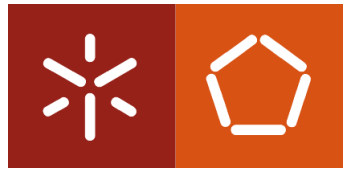
TCP service: serviço de transferência de bytes fiável de um processo para outro



Programação com Sockets: TCP

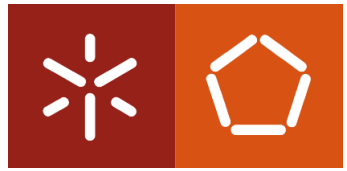


Programação com Sockets: TCP



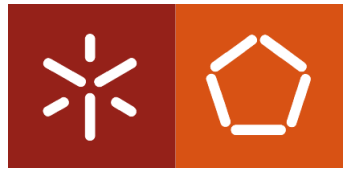
- **O processo cliente tem que executar os seguintes passos:**
  - Estabelecer um *socket*, usando a chamada ao sistema **socket()**
  - Através do *socket* estabelecido conectar-se ao processo servidor, usando a chamada ao sistema **connect()**
  - Enviar e receber dados através do canal estabelecido, usando as chamadas ao sistema **send()** e **recv(),**
  - Libertar o *socket* estabelecido, através da chamada ao sistema **close()**

Programação com Sockets: TCP



- **O processo servidor tem que executar os seguintes passos:**
  - Estabelecer um *socket*, usando a chamada ao sistema **socket()**
  - Associar o *socket* criado a uma identificação (constituída por endereço IP da máquina onde reside o servidor e número de porta associado à aplicação em causa), usando a chamada ao sistema **bind()**
  - Ficar à escuta de pedido de ligação, usando a chamada ao sistema **listen()**
  - Aceitar um pedido de ligação através da chamada ao sistema **accept()**
  - Enviar e receber dados através do canal estabelecido, usando as chamadas ao sistema **send()** e **recv()**
  - Libertar o *socket* estabelecido, através da chamada ao sistema **close()**

Exemplo Java: Servidor TCP



```
class TCPServer {
  public static void main(String args[]) throws Exception
    {
      ServerSocket  welcomeSocket = new ServerSocket(9876);

      while(true)  {
        Socket socketPedido =  welcomeSocket.accept();
        BufferedReader in =
              new BufferedReader(new InputStreamReader(socketPedido.getInputStream()));
         PrintWriter out  = new PrintWriter(socketPedido.getOutputStream(), true);

        String pedido = in.readLine();
        String resposta = pedido.toUpperCase();
        out.println(resposta);
        socketPedido.close();
       }
    }
}
```

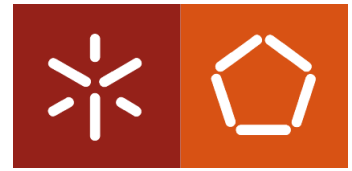
O construtor do objecto faz o equivalente a Socket(), Bind() e Listen()

Espera conexão Accept()

Lê pedido e envia resposta

Fecha conexão...

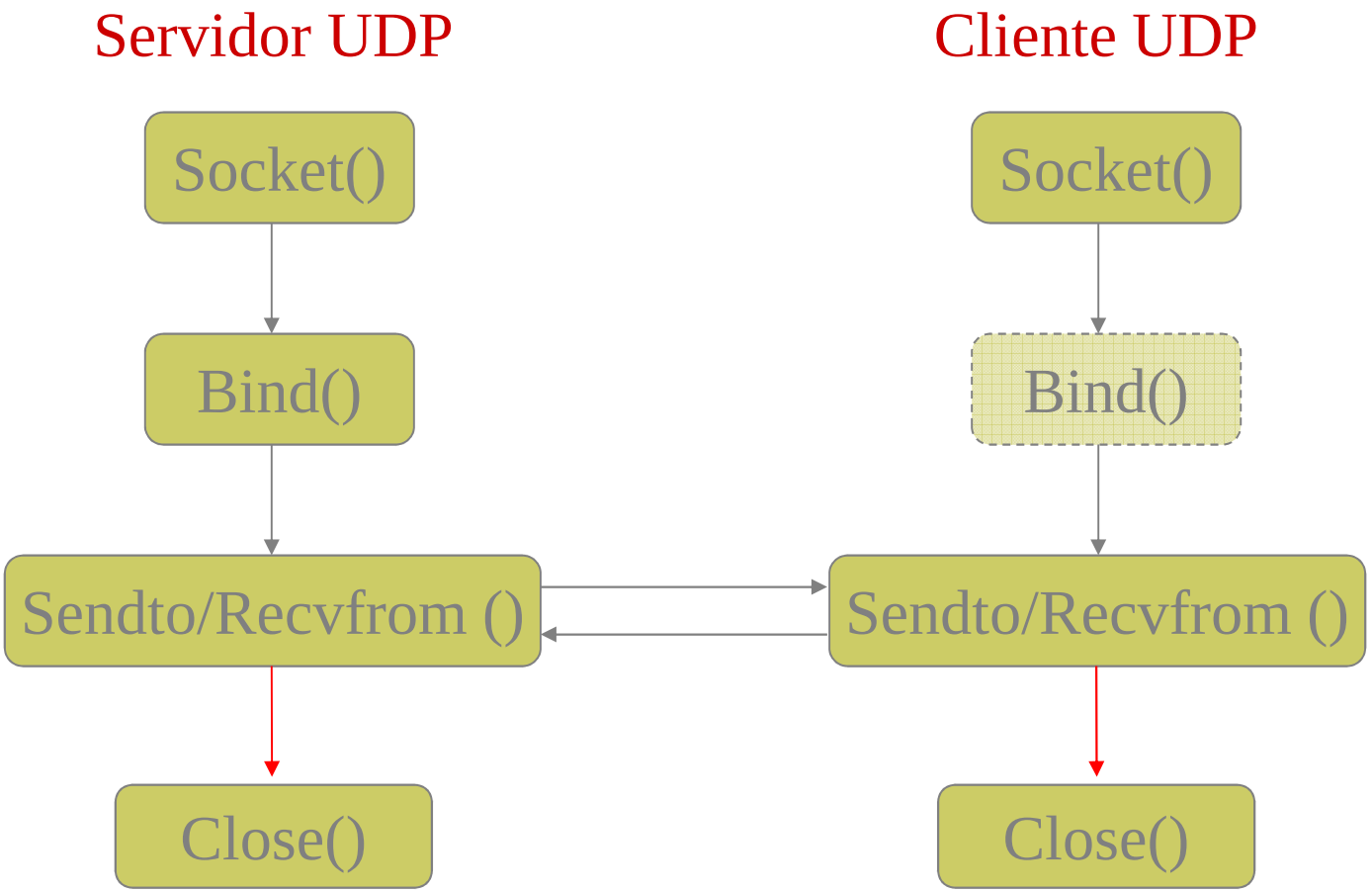
Exemplo Java: Ciente TCP



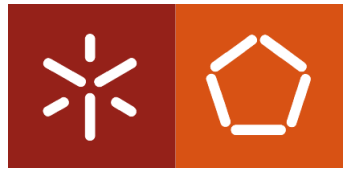
```
class TCPClient {
   public static void main(String args[]) throws Exception
   {
      InetAddress serverAddress = InetAddress.getByName("localhost");
      Socket socket = new Socket(serverAddress, 9876);

      BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      PrintWriter out  = new PrintWriter(socket.getOutputStream(), true);

      String pedido = "Pedido ao servidor...";
       out.println(pedido);
      String resposta = in.readLine();
      System.out.println("Resposta:" + resposta);
      socket.close();
     }
}
```

O construtor faz Socket() e Connect()

Programação com Sockets: UDP

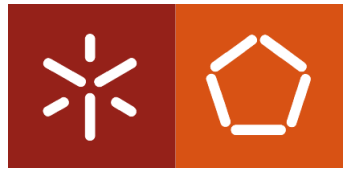


Programação com Sockets: UDP



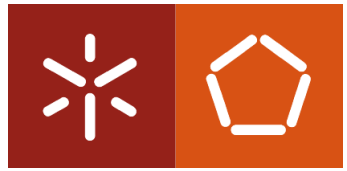
- **Para ligações em que tipicamente as mensagens são independentes umas das outras, e não é necessário um serviço de transporte fiável utiliza-se o protocolo de transporte UDP e sockets do tipo Datagrama (**SOCK_DGRAM**)**
- **Neste caso usam-se chamadas ao sistema diferentes das usadas nas ligações TCP**
  - **sendto**() e **receivefrom**() para enviar e receber mensagens respectivamente

Exemplo Java: Servidor UDP



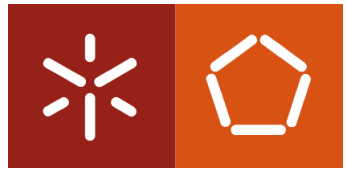
```
class UDPServer {
  public static void main(String args[]) throws Exception
    {
       DatagramSocket s= new DatagramSocket(9876);
       byte[] aReceber = new byte[1024];

       while(true)  {
         DatagramPacket pedido =  new DatagramPacket(aReceber, aReceber.length);
         s.receive(pedido);
         String pedidoString = new String(pedido.getData(), 0, pedido.getLength());

         InetAddress IPAddress = pedido.getAddress();
         int porta = pedido.getPort();
         byte[] aEnviar= pedidoString.toUpperCase().getBytes();
         DatagramPacket resposta = new DatagramPacket(aEnviar, aEnviar.length, IPAddress, porta);
         s.send(resposta);
        }
    }
}
```

Exemplo Java: Ciente UDP



```
class UDPClient {
  public static void main(String args[]) throws Exception
  {
     InetAddress IPAddress = InetAddress.getByName("localhost");
     DatagramSocket s = new DatagramSocket();

     byte[] aEnviar = new String("Pedido ao servidor....").getBytes();
     DatagramPacket p = new DatagramPacket(aEnviar, aEnviar.length, IPAddress, 9876);
     s.send(p);

     byte[] aReceber = new byte[1024];
     DatagramPacket r = new DatagramPacket(aReceber, aReceber.length);
     s.receive(r);

     String resposta = new  String(r.getData(),0, r.getLength());
     System.out.println("Resposta:" + resposta);
     s.close();
    }
}
```

Threads em Java: Servidor TCP

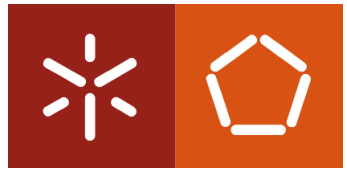


```
public class AtendePedidoTCP extends Thread {

    private BufferedReader in;
    private PrintWriter out;

    public AtendePedidoTCP(Socket s) throws IOException {
        this.in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        this.out = new PrintWriter(s.getOutputStream(), true);
    }

    public void run() {
        String pedido, resposta;
        try {
            pedido = in.readLine();
            resposta = pedido.toUpperCase();
            out.println(resposta);
        } catch (IOException ex) {
            ...
        }
    }
}
```

Subclasse de Thread

Reescreve metódo RUN
(trata apenas um pedido)

Threads em Java: Servidor TCP

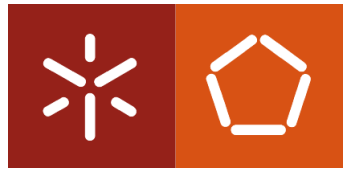


```
public class AceitaPedidosTCP {
   private int port = 6789;  // default

   public AceitaPedidosTCP(int port) {
      this.port = port;
   }

   public AceitaPedidosTCP() {
   }

   public void Atendimento() throws IOException {
    ServerSocket welcomeSocket = new ServerSocket(this.port);
    while(true) {
       Socket connectionSocket = welcomeSocket.accept();
       AtendePedidoTCP atendedor = new AtendePedidoTCP(connectionSocket);
       atendedor.start();
    }
   }
}
```

Classe Principal
(aceite pedidos e cria threads)

Aceita pedido
Cria thread (por pedido)
Inicia a thread

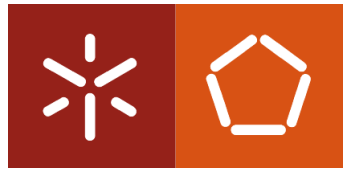
Threads em Java: Servidor TCP



```
public static void main(String[] args) {
    try {
        AceitaPedidosTCP server = new AceitaPedidosTCP(6789);
        server.Atendimento();
    } catch (IOException ex) {
        ....
    }
}
```

Programa Principal
(cria servidor de atendimento)

Threads em Java: Servidor TCP

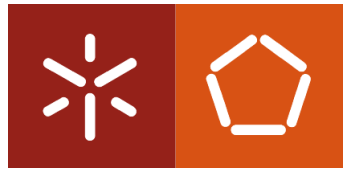


```
public class AtendePedidoTCP implements Runnable {

    private BufferedReader in;
    private PrintWriter out;

    public AtendePedidoTCP(Socket s) throws IOException {
        this.in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        this.out = new PrintWriter(s.getOutputStream(), true);
    }

    public void run() {
        String pedido, resposta;
        try {
            pedido = in.readLine();
            resposta = pedido.toUpperCase();
            out.println(resposta);
        } catch (IOException ex) {
            ...
        }
    }
}
```

Implementa interface Runnable

Escreve metódo RUN (igual ao anterior)

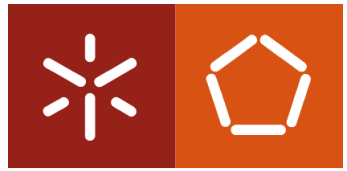
Threads em Java: Servidor TCP



```
public class AceitaPedidosTCP {
   private int port = 6789;  // default

   public AceitaPedidosTCP(int port) {
      this.port = port;
   }

   public AceitaPedidosTCP() {
   }

   public void Atendimento() throws IOException {
    ServerSocket welcomeSocket = new ServerSocket(this.port);
    while(true) {
       Socket connectionSocket = welcomeSocket.accept();
       AtendePedidoTCP atendedor = new AtendePedidoTCP(connectionSocket);
       Thread worker = new Thread(atendedor);
       worker.start()
    }
   }
}
```

Só muda a forma de criar as threads...

Sockets UDP Multicast



- **Caso particular de um socket UDP... o destino é um grupo e não um host! Servidor que pretende receber pacotes deve fazer previamante "join" ao grupo**

```
// join a Multicast group and send the group salutations ...
      String msg = "Hello";
      InetAddress group = InetAddress.getByName("228.5.6.7");
      MulticastSocket s = new MulticastSocket(6789);
      s.joinGroup(group);
      DatagramPacket hi = new DatagramPacket(msg.getBytes(), msg.length(), group, 6789);
      s.send(hi);
// get their responses!
      byte[] buf = new byte[1000];
      DatagramPacket recv = new DatagramPacket(buf, buf.length);
      s.receive(recv); ...
// OK, I'm done talking - leave the group...
      s.leaveGroup(group);
```