UNIVERSIDADE FEDERAL DO CEARÁ
Campus Quixadá
Disciplina: Sistemas Distribuídos

Aula 3 – Comunicação entre processos

Prof. Carlos Bruno

Agenda

- Características da comunição entre processos;
- *Sockets;*
- Datagrama UDP;
- Fluxo TCP;
- Representação externa de dados

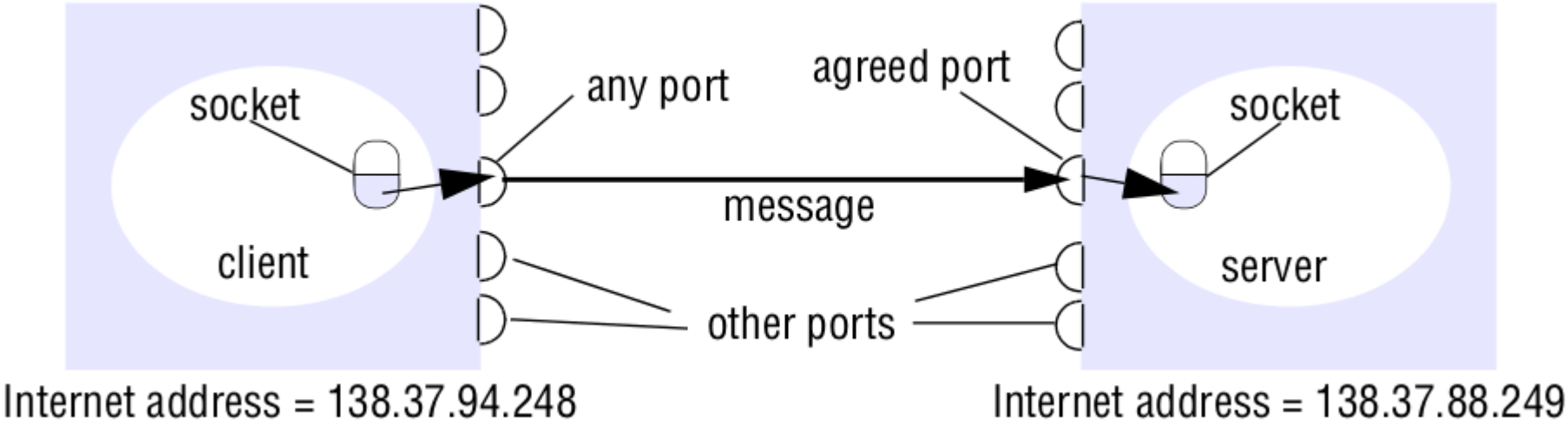
Características

- Síncrona ou assíncrona;
- Destino das mensagens
  - IP e porta
- Confiabilidade
  - TCP
  - UDP
- Ordem das mensagens

Sockets

- Uma abstração de endpoints para comunicação entre processos;
- Comunicação entre processos consiste no envio de mensagens entre *sockets*;
- Cada *socket* deve estar associado a um número de porta e a um número IP;
- Há 65.536 portas, das quais 1.024 são reservadas/bem conhecidas;
- Um socket pode ser UDP ou TCP.

Sockets


socket
any port
agreed port
socket
message
client
server
other ports
Internet address = 138.37.94.248
Internet address = 138.37.88.249

Sockets

- **InetAddress**, classe da API Java que representa endereços IP
  - InetAddress aComputer = InetAddress.getByName("bruno.dcs.qmul.ac.uk");
- O método estático *getByName* encapsula as mensagens enviadas a um Sistema de Nomes de Domínio (**DNS**);
- *UnknownHostException*

Datagrama UDP

- Sem confirmação e sem retransmissão de datagramas;
- Tamanho das mensagens: 8 KB
- Mensagens podem ser descartadas por não haver checagem de erros e/ou buffer disponível;
- Datagramas podem ser entregues fora de ordem;
- A própria aplicação deve fornecer mecanismos de checagem de erros;
- DNS, VOIP, DHCP.

Datagrama UDP (JAVA API)

- **DatagramPacket** *(mensagem, tamanho da msg, IP, portaServidor)*; datagrama que será enviado;
- **DatagramPacket** *(mensagem, tamanho da msg)*; datagrama que será recebido;
- **getData()**: extrai os dados do datagrama;
- **getPort()**: número da porta;
- **getAddress()**: número IP.

Datagrama UDP (JAVA API)

- **DatagramSocket** *(porta)*; envia e recebe datagramas UDP; *porta é opcional;
- **SocketException: se a porta escolhida estiver em uso;**
- **getData()**: extrai os dados do datagrama;
- **getPort()**: número da porta;
- **setSoTimeout()**: tempo de espera antes de lançar uma *InterruptedIOException.*

Datagrama UDP (JAVA API)

```
public class UDPClient{
   public static void main(String args[]){
            DatagramSocket aSocket = null;
                        aSocket = new DatagramSocket();
                        byte [] m = args[0].getBytes();
                        InetAddress aHost = InetAddress.getByName(args[1]);
                        int serverPort = 6789;
                        DatagramPacket request = new DatagramPacket(m,  m.length(), aHost, serverPort);
                        aSocket.send(request);
                        byte[] buffer = new byte[1000];
                        DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                        aSocket.receive(reply);
                        System.out.println("Reply: " + new String(reply.getData()));
             }

}
```

Datagrama UDP (JAVA API)

```
public class UDPServer{
        public static void main(String args[]){
        DatagramSocket aSocket = null;
                aSocket = new DatagramSocket(6789);
                byte[] buffer = new byte[1000];
                while(true){
                   DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                  aSocket.receive(request);
                  DatagramPacket reply = new DatagramPacket(request.getData(),
                          request.getLength(), request.getAddress(), request.getPort());
                  aSocket.send(reply);
        }
}
```

Fluxo TCP

- Confirmação e retransmissão de pacotes;
- Tamanho das mensagens: a aplicação decide;
- Controle de fluxo;
- Pacotes entregues em ordem e sem duplicações;
- Os processos estabelecem uma conexão antes de escrever e ler no fluxo tcp;
- O TCP pode gerar overhead para uma aplicação simples;
- O fluxo deve respeitar uma ordem de escrita e leitura. Ex: double → int → double.
- HTTP, FTP, Telnet, SMTP

Fluxo TCP (Java API)

- **ServerSocket**: cria um socket em alguma porta no servidor; aguarda (escuta) até que conexão seja estabelecida;
- **Socket**: usada por um par de processos. Recebe um nome de host e um número de porta.
  - **getInputStream()**
  - **getOutputStream()**

Cliente TCP

```
public class TCPClient {
        public static void main (String args[]) {
        // arguments supply message and hostname of destination
        Socket s = null;
                int serverPort = 7896;
                s = new Socket(args[1], serverPort);
                DataInputStream in = new DataInputStream( s.getInputStream());
                DataOutputStream out =
                        new DataOutputStream( s.getOutputStream());
                out.writeUTF(args[0]);          // UTF is a string encoding see Sn 4.3
                String data = in.readUTF();
                System.out.println("Received: "+ data) ;

        }
}
```

Servidor TCP

```
public class TCPServer {
   public static void main (String args[]) {
                                int serverPort = 7896;
                                ServerSocket listenSocket = new ServerSocket(serverPort);
                                while(true) {
                                    Socket clientSocket = listenSocket.accept();
                                    Connection c = new Connection(clientSocket);
                                }
                    }
}
class Connection extends Thread {
                DataInputStream in;
                DataOutputStream out;
                Socket clientSocket;
                public Connection (Socket aClientSocket) {
                                clientSocket = aClientSocket;
                                in = new DataInputStream( clientSocket.getInputStream());
                                out =new DataOutputStream( clientSocket.getOutputStream());
                                this.start();
                                }
                public void run(){
                                // an echo server
                                String data = in.readUTF();
                                out.writeUTF(data);
                }
}
```

Representação externa

- As estruturas de dados devem ser representadas por sequências de *bytes*;
  - *Nem todas as máquinas possuem a mesma arquitetura;*
  - *Pode haver divergência na representação de dados primitivos;*
  - *A quantidade de bits para representar um inteiro pode variar, mesmo usando-se a mesma linguagem (ex: C);*
  - *ASCII x Unicode*

Representação externa

- Como possibilitar a troca de informações nesse contexto?
  - Os valores podem ser convertidos para um formato externo acordado antes da transmissão e convertidos para a forma local quando recebidos
- Serialização de objetos no Java;
- XML (Extensible Markup Language);
- JSON (JavaScript Object Notation);
- Protocol Buffers (Google).

Java Object Serialization

```
public class Person implements Serializable {
    private String name;
    private String place;
    private int year;
    public Person(String aName, String aPlace, int aYear) {
        name = aName;
        place = aPlace;
        year = aYear;
   }
   // followed by methods for accessing the instance
variables
}
```

XML

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

XML

```
<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>
```

JSON

{ "name":"Nayra", "place":”itapiúna”, "year":2018 }