Computación en Internet I

Andrés A. Aristizábal P.
aaaristizabal@icesi.edu.co

Departamento de Tecnologías de Información y Comunicaciones

UNIVERSIDAD ICESI

2023-1

Agenda

1. **Java Socket Programming**
   - Introduction
   - Types of sockets
   - The Connectionless Datagram Socket
   - Examples
   - Exercises

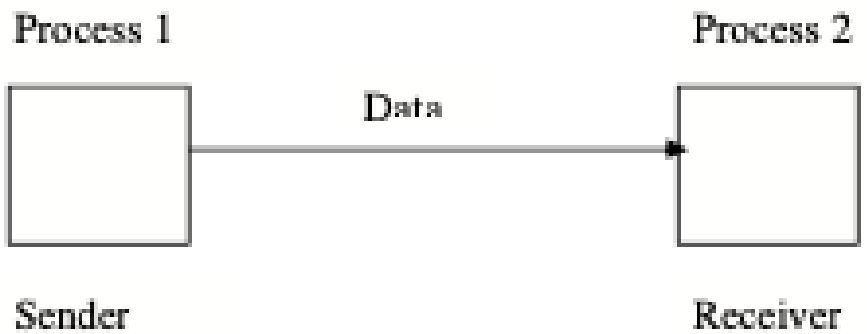
Agenda del día

1. **Java Socket Programming**
   - Introduction
   - Types of sockets
   - The Connectionless Datagram Socket
   - Examples
   - Exercises

What is the interprocess communication (IPC)?

What is the interprocess communication (IPC)?

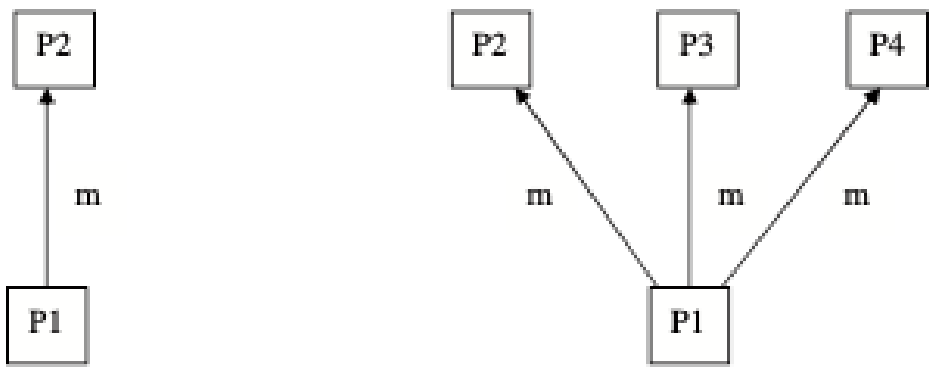
- The backbone of distributed computing.
- Ability for separate, independent processes to communicate among themselves to collaborate on a task.



How does this communication take place?

How does this communication take place?

- Two or more processes engage in a protocol.
- A process can be a sender at some instant of the communication.
- Can be a receiver of the data at another instant of the communication.
- Unicast: data is sent from one process to another single process.
- Multicast: data is sent from one process to more than one process at the same time.



Which facilities do OS provide for IPC?

Which facilities do OS provide for IPC?

- Message queues.
- Semaphores.
- Shared memory.

Which facilities do OS provide for IPC?

- Message queues.
- Semaphores.
- Shared memory.

How to spare the programmer with the complexities associated to system-level details?

Which facilities do OS provide for IPC?

- Message queues.
- Semaphores.
- Shared memory.

How to spare the programmer with the complexities associated to system-level details?

- IPC application programming interface (API).
- The Socket API is a low-level programming facility for implementing IPC.
  - Provides a programming construct called a *s*ocket.
  - A process wishing to communicate with another must instantiate a socket.
  - After instantiating sockets processes may send and receive data by using operations provided by the API.

How are sockets identified?

How are sockets identified?

- Uniquely identified by the IP address and the port number at which the socket is opened.
  - Port numbers can be at most 66535.
  - Well-known processes have their sockets opened on dedicated port numbers ≤ 1024.
  - User-defined processes have to be run on port numbers > 1024.

Agenda del día

1. **Java Socket Programming**
   - Introduction
   - Types of sockets
   - The Connectionless Datagram Socket
   - Examples
   - Exercises

Types of sockets

Which types of sockets do we have?

Types of sockets

Which types of sockets do we have?

- Datagram sockets (use UDP)
  - Transports packets in a connectionless manner.
  - Each datagram is addressed and routed individually.
  - It may arrive at the receiver in any order.
- Stream sockets (use TCP)
  - It is connection-oriented.
  - Transports a stream of data over a logical connection established between sender and receiver.
  - Data sent from a sender to a receiver is guaranteed to be received in the order they were sent.

Agenda del día

1. **Java Socket Programming**
   - Introduction
   - Types of sockets
   - The Connectionless Datagram Socket
   - Examples
   - Exercises

The Connectionless Datagram Socket

Which classes are provided for the datagram socket API?

Which classes are provided for the datagram socket API?

- The DatagramSocket class for the sockets.
- The DatagramPacket class for the packets exchanged.

The Connectionless Datagram Socket

Which classes are provided for the datagram socket API?

- The DatagramSocket class for the sockets.
- The DatagramPacket class for the packets exchanged.

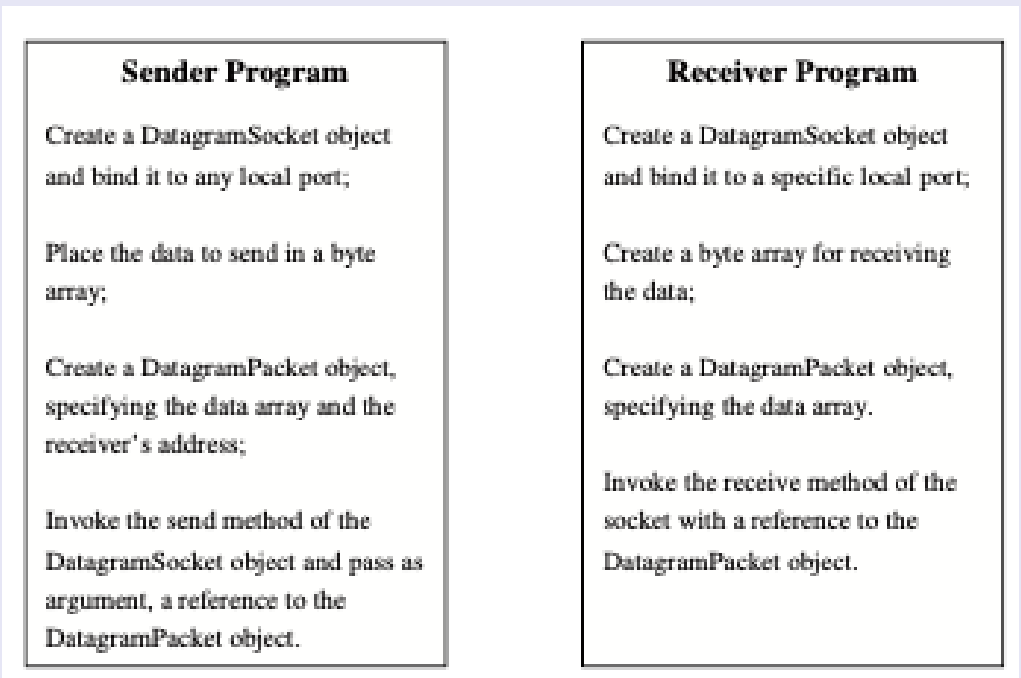
What are the steps to be followed?

The Connectionless Datagram Socket

Which classes are provided for the datagram socket API?

- The DatagramSocket class for the sockets.
- The DatagramPacket class for the packets exchanged.

What are the steps to be followed?



Which are the key methods for the DatagramSocket class?

Which are the key methods for the DatagramSocket class?

- DatagramSocket()
  - Constructs objects and binds it to any available port on the local host machine.
- DatagramSocket(int port)
  - Constructs object and binds it to the specified port on the local host machine.
- DatagramSocket(int port, InetAddress laddr)
  - Constructs object and binds it to the specified local address and port.
- void close()
  - Closes the datagram socket.

Which are the key methods for the DatagramSocket class?

- void connect(InetAddress address, int port)
  - Connects the datagram socket to the specified remote address and port number on the machine with that address.
- InetAddress getLocalAddress()
  - Returns the local InetAddress to which the socket is connected.
- int getLocalPort()
  - Returns the port number on the local host to which the datagram socket is bound.
- InetAddress getInetAddress()
  - Returns the IP address to which the datagram socket is connected to at the remote side.

Which are the key methods for the DatagramSocket class?

- int getPort()
  - Returns the port number at the remote side of the socket.
- void receive(DatagramPacket packet)
  - Receives a datagram packet object from this socket.
- void send(DatagramPacket packet)
  - Sends a datagram packet object from this socket.
- void setSoTimeout(int timeout)
  - Set the timeout value for the socket, in milliseconds.

Which are the key methods for the DatagramPacket class?

Which are the key methods for the DatagramPacket class?

- DatagramPacket(byte[] buf, int length, InetAddress, int port)
  - Constructs a datagram packet object with the contents stored in a byte array, buf, of specified length to a machine with the specified IP address and port number.
- InetAddress getAddress()
  - Returns the IP address of the machine at the remote side to which the datagram is being sent or from which the datagram was received.
- byte [] getData()
  - Returns the data buffer stored in the packet as a byte array.
- int getLength()
  - Returns the length of the data buffer in the datagram packet sent or received.

Which are the key methods for the DatagramPacket class?

- int getPort()
  - Returns the port number to which the datagram socket is bound to which the datagram is being sent or from which the datagram is received.
- void setData(byte [])
  - Sets the data buffer for the datagram packet.
- void setAddress(InetAddress iaddr)
  - Sets the datagram packet with the IP address of the remote machine to which the packet is being sent.
- void setPort(int port)
  - Sets the datagram packet with the port number of the datagram socket at the remote host to which the packet is sent.

The Connectionless Datagram Socket

To take into account

- With connectionless sockets, a DatagramSocket object bound to a process can be used to send datagrams to different destinations.
- Multiple processes can simultaneously send datagrams to the same socket bound to a receiving process.
- The send operations are non-blocking and the receive operations are blocking.

Agenda del día

1. **Java Socket Programming**
   - Introduction
   - Types of sockets
   - The Connectionless Datagram Socket
   - Examples
   - Exercises

Program to receive-send a single Datagram

- Server:
  - Defines a maximum size for the received message.
  - Gets the local port number as a parameter.
  - Creates a socket associated to the local port.
  - Creates a buffer of bytes where to receive.
  - Creates a packet with the buffer and its length.
  - By means of the socket it receives the packet.
  - Gets the info from the buffer and prints it.
  - Closes the socket.

Program to receive-send a single Datagram

- Client:
  - Receives and stores, address, port and message form user.
  - Creates a socket.
  - Creates buffer of bytes and stores the message in it.
  - Creates a packet with the buffer, its size, the address and the port.
  - Sends the packet via the socket.
  - Closes the socket.

Program to receive-send a single Datagram

Server

```
import java.net.*;
import java.io.*;
public class DatagramReceiver{
    public static void main(String[] args) {
        try {
            int MAX_LEN = 40;
            int localPortNum = Integer.parseInt(args[0]);
            DatagramSocket mySocket = new DatagramSocket(localPortNum);
            byte[] buffer = new byte[MAX_LEN];
            DatagramPacket packet = new DatagramPacket(buffer, MAX_LEN);
            mySocket.receive(packet);
            String message = new String(buffer);
            System.out.println(message);
            mySocket.close();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Program to receive-send a single Datagram

Client

```
import java.net.*;
import java.io.*;
public class DatagramSender {
    public static void main(String[] args) {
        try {
            BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Input the address:");
            InetAddress receiverHost = InetAddress.getByName(input.readLine());
            System.out.println("Input the port:");
            int receiverPort = Integer.parseInt(input.readLine());
            System.out.println("Input the message:");
            String message = input.readLine();
            input.close();
            DatagramSocket mySocket = new DatagramSocket();
            byte[] buffer = message.getBytes();
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length, receiverHost, receiverPort);
            mySocket.send(packet);
            mySocket.close();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Program to Send and Receive a Message in both Directions

- Server:
  - Defines a maximum size for the received message.
  - Gets the local port number as a parameter.
  - Creates a socket associated to the local port.
  - Creates a buffer of bytes where to receive.
  - Creates a packet with the buffer and its length.
  - By means of the socket it receives the packet.
  - Gets the info from the buffer and prints it.
  - Creates a buffer of bytes and stores the new message in it.
  - Gets the source address and port from the packet.
  - Constructs a new message.
  - Creates a packet with the buffer, its size, the address and the port.
  - Sends the packet via the socket.
  - Closes the socket.

Program to Send and Receive a Message in both Directions

- Client:
  - Receives and stores, address, port and message form user.
  - Creates a socket.
  - Creates a buffer of bytes and stores the message in it.
  - Creates a packet with the buffer, its size, the address and the port.
  - Sends the packet via the socket.
  - Defines a maximum size for the received message.
  - Creates a buffer of bytes where to receive.
  - Creates a packet with the buffer and its length.
  - By means of the socket it receives the packet.
  - Gets the info from the buffer and prints it.

Program to Send and Receive a Message in both Directions

Server

```
import java.net.*;
import java.io.*;
public class DatagramReceiverSender {
    public static void main(String[] args) {
        try {
            int MAX_LEN = 60;
            int localPortNum = Integer.parseInt(args[0]);
            DatagramSocket mySocket = new DatagramSocket(localPortNum);
            byte[] recvBuffer = new byte[MAX_LEN];
            DatagramPacket packet = new DatagramPacket(recvBuffer, MAX_LEN);
            mySocket.receive(packet);
            String message = new String(recvBuffer);
            System.out.println("\n"+message);
            InetAddress senderAddress = packet.getAddress();
            int senderPort = packet.getPort();
            String messageToSend = "This was the received message: "+message;
            byte[] sendBuffer = messageToSend.getBytes();
            DatagramPacket datagram = new DatagramPacket(sendBuffer, sendBuffer.length, senderAddress, senderPort);
            mySocket.send(datagram);
            mySocket.close();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Program to Send and Receive a Message in both Directions

Client

```
import java.net.*;
import java.io.*;
public class DatagramSenderReceiver {
    public static void main(String[] args) {
        try {
            BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Input the address:");
            InetAddress receiverHost = InetAddress.getByName(input.readLine());
            System.out.println("Input the port:");
            int receiverPort = Integer.parseInt(input.readLine());
            System.out.println("Input the message:");
            String message = input.readLine();
            DatagramSocket mySocket = new DatagramSocket();
            byte[] sendBuffer = message.getBytes();
            DatagramPacket packet = new DatagramPacket(sendBuffer, sendBuffer.length, receiverHost, receiverPort);
            mySocket.send(packet);
            int MESSAGE_LEN = 60;
            byte[] recvBuffer = new byte[MESSAGE_LEN];
            DatagramPacket datagram = new DatagramPacket(recvBuffer, MESSAGE_LEN);
            mySocket.receive(datagram);
            String recvdString = new String(recvBuffer);
            System.out.println("\n"+recvdString);
            mySocket.close();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Program to calculate the factorial of a number

- Server:
  - Creates the input and output buffers of bytes.
  - Creates the socket associated to a port.
  - Creates a datagram by using the buffer.
  - Receives the datagram by means of the socket.
  - Extracts the info from the datagram.
  - Gets the address and port from the datagram.
  - Converts the data as a number.
  - Calculates the factorial.
  - Saves the result in the buffer of bytes.
  - Creates the datagram to be sent.
  - Sends the datagram via the socket.
  - Prints what was sent.

Program to calculate the factorial of a number

- Client:
  - Receives the address as a parameter.
  - Creates the input and output buffers of bytes.
  - Reads the input number from the standard input.
  - Creates a socket.
  - Stores the message in the output buffer of bytes.
  - Creates a packet with the buffer, its size, the address and the port.
  - Sends the packet via the socket.
  - Prints what is sent to the server.
  - Creates a datagram with the input buffer and its length.
  - By means of the socket it receives the packet.
  - Gets the info from the buffer and prints it.

Program to calculate the factorial of a number

Server

```
import java.net.*;
public class FactServer {
    public static void main(String args[]) throws Exception {
        byte[] rbuf = new byte[10];
        byte[] sbuf = new byte[10];
        DatagramSocket socket = new DatagramSocket(5000);
        System.out.println("Server ready");
        DatagramPacket rpkt = new DatagramPacket(rbuf, rbuf.length);
        socket.receive(rpkt);
        String data = new String(rpkt.getData(), 0, rpkt.getLength());
        InetAddress addr = rpkt.getAddress();
        int port = rpkt.getPort();
        int fact = 1;
        int n = Integer.parseInt(data);
        System.out.println("Received: " + n + " from " + addr + ":" + port);
        for (int i = 2; i <= n; i++)
            fact *= i;
        sbuf = String.valueOf(fact).getBytes();
        DatagramPacket spkt = new DatagramPacket(sbuf, sbuf.length, addr, port);
        socket.send(spkt);
        System.out.println("Sent: " + fact);
    }
}
```

Program to calculate the factorial of a number

Client

```
import java.io.*;
import java.net.*;

public class FactClient {
    public static void main(String args[]) throws Exception {
        byte[] rbuf = new byte[1024];
        byte[] sbuf = new byte[1024];
        BufferedReader fromUser = new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket socket = new DatagramSocket();
        InetAddress addr = InetAddress.getByName(args[0]);
        System.out.print("Enter an integer: ");
        String data = fromUser.readLine();
        sbuf = data.getBytes();
        DatagramPacket spkt = new DatagramPacket(sbuf, sbuf.length, addr, 5000);
        socket.send(spkt);
        System.out.println("Sent to server: " + data);
        DatagramPacket rpkt = new DatagramPacket(rbuf, rbuf.length);
        socket.receive(rpkt);
        data = new String(rpkt.getData(), 0, rpkt.getLength());
        System.out.println("Received from server: " + data);
        socket.close();
    }
}
```

Program to calculate the factorial of a number

Handler

```
import java.io.*;
import java.net.*;
public class Handler extends Thread {
    DatagramSocket socket;
    DatagramPacket pkt;
    public Handler(DatagramSocket socket) {
        byte[] rbuf = new byte[10];
        this.socket = socket;
        this.pkt = new DatagramPacket(rbuf, rbuf.length);
    }
```

Program to calculate the factorial of a number

Handler

```
    public void run() {
        try {
         while (true) {
              socket.receive(pkt);
              byte[] sbuf = new byte[10];
              String data = new String(pkt.getData(), 0, pkt.getLength());
              InetAddress addr = pkt.getAddress();
               int port = pkt.getPort();
               int fact = 1;
               int n = Integer.parseInt(data);
              System.out.println("Received: " + n + " from " + addr + ":" + port);
              for (int i = 2; i <= n; i++)
                    fact *= i;
              sbuf = String.valueOf(fact).getBytes();
              DatagramPacket spkt = new DatagramPacket(sbuf, sbuf.length, addr, port);
              socket.send(spkt);
              System.out.println("Sent: " + fact);
              socket.close();
          }
        }
        catch (IOException e) {
        }
    }
}
```

Program to calculate the factorial of a number

Server

```
import java.io.*;
import java.net.*;
public class FactServerT {
    public static void main(String args[]) throws Exception {
            DatagramSocket socket = new DatagramSocket(5000);
            System.out.println("Server ready");
            Thread myThread = new Handler(socket);
            myThread.start();
    }
}
```

Agenda del día

1. **Java Socket Programming**
   - Introduction
   - Types of sockets
   - The Connectionless Datagram Socket
   - Examples
   - Exercises

Exercises

1. Run the previous examples.
2. Develop a simple chatting application using Connectionless sockets.
   - When the user presses the enter key, whatever characters have been typed by the user until then, are transferred to the other end.
   - You can also assume that for every message entered from one end, a reply must come from the other end, before another message could be sent.
   - More than one message cannot be sent from a side before receiving a response from the other side.
   - Assume the maximum number of characters that can be transferred in a message to be 1000. The chat will be stopped by pressing Ctrl+C on both sides.