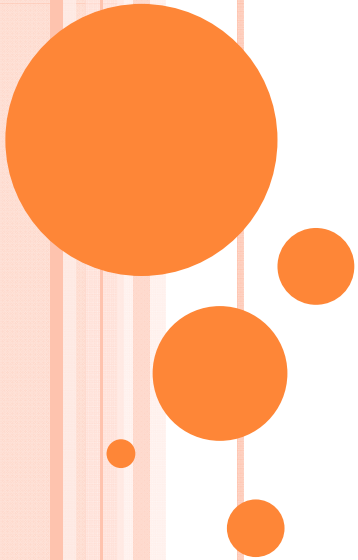
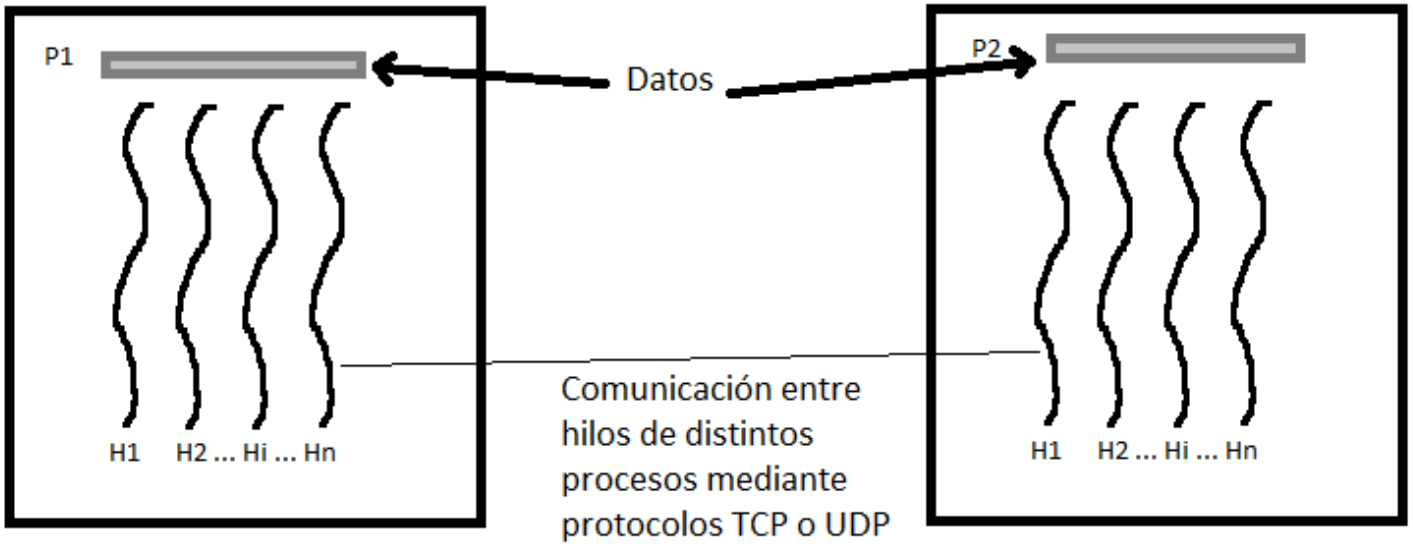


Comunicación y sincronización entre procesos

Germán Lescano

gelescano@unse.edu.ar | german.lescano@gmail.com

Comunicación y sincronización entre procesos



SensorEntrada (Servidor)

```
1.  public class SensorEntrada {
2.
3.      public static void main(String[] args){
4.          try {
5.              Contacto contacto = new Contacto();
6.              SimuladorContacto simulador = new SimuladorContacto(contacto);
7.              ServerSocket socketSensor = new ServerSocket(3386);
8.              while(true){
9.                  Socket socketCliente = socketSensor.accept();
10.                 Conexion cnxCliente = new Conexion(socketCliente, contacto);

11.             }
12.         } catch (IOException ex) {
13.             Logger.getLogger(SensorEntrada.class.getName()).log(Level.SEVERE,
     null, ex);
14.         }
15.     }
16.
17. }
```

SensorEntrada (Servidor) (Cont.)

```
1.  class Conexion extends Thread{
2.      Socket cnxCliente;
3.      DataOutputStream salida;
4.      Contacto contacto;
5.
6.      public Conexion(Socket cnx, Contacto c) {
7.          try {
8.              cnxCliente = cnx;
9.              salida = new DataOutputStream(cnxCliente.getOutputStre
    am());
10.             contacto = c;
11.             start();
12.         } catch (IOException ex) {
13.             Logger.getLogger(Conexion.class.getName()).log(Level.S
    EVERE, null, ex);
14.         }
15.     }
16.
```

SensorEntrada (Servidor) (Cont. Clase Conexion)

```
1.      @Override
2.      public void run() {
3.          while (true) {
4.              try {
5.                  contacto.EsperarContacto();
6.                  salida.writeShort(1);
7.              } catch (IOException ex) {
8.                  Logger.getLogger(Conexion.cla
  ss.getName()).log(Level.SEVERE, null, ex);
9.              }
10.         }
11.     }
12.}
```

SensorEntrada (Servidor)

```
1.  public class SimuladorContacto extends Thread {
2.
3.      Contacto contacto;
4.
5.      public SimuladorContacto(Contacto c){
6.          contacto = c;
7.          start();
8.      }
9.
10.     @Override
11.     public void run(){
12.         Scanner entrada = new Scanner(System.in);
13.         while(true){
14.             System.out.println("Ingrese 1 para representar un contacto");
15.             short ingreso = entrada.nextShort();
16.             if (ingreso == 1){
17.                 // Señalar ingreso
18.                 contacto.IndicarContacto();
19.             }
20.         }
21.     }
22.
23. }
```

SensorEntrada (Servidor)

```
1.  public class Contacto {
2.
3.      boolean enContacto;
4.
5.      public Contacto(){
6.          enContacto = false;
7.      }
8.
9.      public synchronized void EsperarContacto(){
10.         if (!enContacto){
11.             try {
12.                 wait();
13.             } catch (InterruptedException ex) {
14.                 Logger.getLogger(Contacto.class.getName()).log(Level.SEVERE,
    null, ex);
15.             }
16.         }
17.         enContacto = false;
18.         notifyAll();
19.     }
```

SensorEntrada (Servidor)

```
1.   public synchronized void IndicarContacto(){
2.         if (enContacto){
3.             try {
4.                 wait();
5.             } catch (InterruptedException ex) {
6.                 Logger.getLogger(Contacto.class.ge
  tName()).log(Level.SEVERE, null, ex);
7.             }
8.         }
9.
10.        enContacto = true;
11.        notifyAll();
12.    }
13.}
```

Cliente Sensor Entrada

```
1. public class ClienteSensorEntrada {
2.
3.     public static void main(String[] args) {
4.         ComunicacionSensorEntrada objSensorEntrada
   = new ComunicacionSensorEntrada();
5.     }
6.
7. }
```

Cliente Sensor Entrada

```
1.  class ComunicacionSensorEntrada extends Thread {
2.
3.      InetAddress direccionSensorEntrada;
4.      Socket cnxSensorEntrada;
5.      DataInputStream entrada;
6.
7.      public ComunicacionSensorEntrada() {
8.          try {
9.              direccionSensorEntrada = InetAddress.getByName("192.168.0.15");
10.             cnxSensorEntrada = new Socket(direccionSensorEntrada, 3386);
11.             entrada = new DataInputStream(cnxSensorEntrada.getInputStream());
12.             start();
13.         } catch (UnknownHostException ex) {
14.             Logger.getLogger(ComunicacionSensorEntrada.class.getName()).log(Level.SEVERE, n
    ull, ex);
15.         } catch (IOException ex) {
16.             Logger.getLogger(ComunicacionSensorEntrada.class.getName()).log(Level.SEVERE, n
    ull, ex);
17.         }
18.
19.     }
20.
```

Cliente Sensor Entrada. Continuación Clase ComunicacionSensorEntrada

```
1.      @Override
2.      public void run() {
3.          while (true) {
4.              try {
5.                  short detectaEntradaAutomovil = entrada.re
adShort();
6.                  if (detectaEntradaAutomovil == 1) {
7.                      System.out.println("Ingreso un automov
il");
8.                  }
9.              } catch (IOException ex) {
10.                 Logger.getLogger(ComunicacionSensorEntrada
.class.getName()).log(Level.SEVERE, null, ex);
11.             }
12.         }
13.     }
14. }
```

Sensor Entrada versión RMI

```
1. public interface ISensorEntrada extends R
   emote {
2.     public void Suscribirse(IClienteSenso
   rEntrada cliente) throws RemoteException;

3. }
```

Sensor Entrada versión RMI

```
1.  public class SensorEntrada extends UnicastRemoteObject implements ISensorEntrada {
2.
3.      ArrayList<IClienteSensorEntrada> suscriptos;
4.      Contacto contacto;
5.      Notificacion hNotificacion;
6.      SimuladorContacto simulador;
7.
8.      public SensorEntrada() throws RemoteException{
9.          super();
10.         contacto = new Contacto();
11.         suscriptos = new ArrayList<>();
12.         simulador = new SimuladorContacto(contacto);
13.         hNotificacion = new Notificacion(contacto, suscriptos);
14.     }
15.
16.     @Override
17.     public void Suscribirse(IClienteSensorEntrada cliente) throws RemoteException {
18.         suscriptos.add(cliente);
19.     }
20.
21. }
```

Sensor Entrada versión RMI

```
1.  class Notificacion extends Thread{
2.      Contacto contacto;
3.      ArrayList<IClienteSensorEntrada> suscriptos;
4.
5.      public Notificacion(Contacto c, ArrayList<IClienteSensorEntrada> s){
6.          contacto = c;
7.          suscriptos = s;
8.          start();
9.      }
10.
11.     @Override
12.     public void run(){
13.         while(true){
14.             contacto.EsperarContacto();
15.             for (int i=0; i < suscriptos.size(); i++){
16.                 try {
17.                     suscriptos.get(i).RecibirNotificacionEvento();
18.                 } catch (RemoteException ex) {
19.                     Logger.getLogger(Notificacion.class.getName()).log(Level.SEVERE, null, ex);
20.                 }
21.             }
22.         }
23.     }
24. }
```

Sensor Entrada versión RMI

```
1.  class SimuladorContacto extends Thread{
2.      Contacto contacto;
3.
4.      public SimuladorContacto(Contacto c){
5.          contacto = c;
6.          start();
7.      }
8.
9.      @Override
10.     public void run(){
11.         Scanner entrada = new Scanner(System.in);
12.         while(true){
13.             System.out.println("Ingrese 1 para señalar contacto");
14.             short c = entrada.nextShort();
15.             if (c == 1){
16.                 contacto.NotificarContacto();
17.             }
18.         }
19.     }
20. }
```

Sensor Entrada versión RMI

```
1.  class Contacto{
2.      boolean hayContacto;
3.
4.      public Contacto(){
5.          hayContacto = false;
6.      }
7.
8.      public synchronized void EsperarContacto(){
9.          if (!hayContacto){
10.             try {
11.                 wait();
12.             } catch (InterruptedException ex) {
13.                 Logger.getLogger(Contacto.class.getName()).log(Level.SEVERE,
    null, ex);
14.             }
15.         }
16.         hayContacto = false;
17.         notifyAll();
18.     }
19.
```

Sensor Entrada versión RMI (Cont. Clase contacto)

```
1.      public synchronized void NotificarContacto(){

2.          if (hayContacto){
3.              try {
4.                  wait();
5.              } catch (InterruptedException ex) {
6.                  Logger.getLogger(Contacto.class.ge
  tName()).log(Level.SEVERE, null, ex);
7.              }
8.          }
9.          hayContacto = true;
10.         notifyAll();
11.     }
12.}
```

Sensor Entrada versión RMI

```
1.  public class ServidorSensorEntradaRMI {
2.
3.      /**
4.       * @param args the command line arguments
5.       */
6.      public static void main(String[] args) {
7.          try {
8.              LocateRegistry.createRegistry(1099);
9.          } catch (RemoteException ex) {
10.             Logger.getLogger(ServidorSensorEntradaRMI.class.getName()).log(Level.SEVERE, null, ex);
11.         }
12.
13.
14.         try {
15.             ISensorEntrada objSensorEntrada;
16.             objSensorEntrada = new SensorEntrada();
17.             Naming.rebind("rmi://localhost:1099/sensorEntrada", objSensorEntrada);
18.         } catch (RemoteException ex) {
19.             Logger.getLogger(ServidorSensorEntradaRMI.class.getName()).log(Level.SEVERE, null, ex);
20.         } catch (MalformedURLException ex) {
21.             Logger.getLogger(ServidorSensorEntradaRMI.class.getName()).log(Level.SEVERE, null, ex);
22.         }
23.
24.     }
25.
26. }
```

Cliente Sensor entrada (Versión RMI)

```
1. public interface IClienteSensorEntrada ex
   tends Remote {
2.     public void RecibirNotificacionEvento
   () throws RemoteException;
3. }
```

Cliente Sensor entrada (Versión RMI)

```
1. public class ClienteSensorEntrada extends UnicastRemoteObject impl
   ements IClienteSensorEntrada, Serializable {
2.
3.     public ClienteSensorEntrada() throws RemoteException{
4.         super();
5.     }
6.
7.     @Override
8.     public void RecibirNotificacionEvento() throws RemoteException
    {
9.         System.out.println("Ingreso un automovil");
10.    }
11.
12.    @Override
13.    public String toString(){
14.        return "ClienteSensorEntrada";
15.    }
16.
17. }
```

Cliente Sensor entrada (Versión RMI)

```
1.      public static void main(String[] args) {
2.              ISensorEntrada objSensorEntrada = null;
3.              IClienteSensorEntrada objClienteSensorEntrada = null;
4.
5.              try {
6.                  LocateRegistry.getRegistry();
7.                  //LocateRegistry.createRegistry(1099);
8.              } catch (RemoteException ex) {
9.                  Logger.getLogger(ClienteSensorEntradaRMI.class.getName()).log(Level.SEVERE, null, ex);
10.             }
11.
12.             try {
13.                 objClienteSensorEntrada = new ClienteSensorEntrada();
14.                 Naming.rebind("rmi://localhost:1099/clienteSE", objClienteSensorEntrada);
15.                 objSensorEntrada = (ISensorEntrada) Naming.lookup("rmi://localhost:1099/sensorEntrada");
16.             } catch (RemoteException ex) {
17.                 Logger.getLogger(ClienteSensorEntradaRMI.class.getName()).log(Level.SEVERE, null, ex);
18.             } catch (MalformedURLException ex) {
19.                 Logger.getLogger(ClienteSensorEntradaRMI.class.getName()).log(Level.SEVERE, null, ex);
20.             } catch (NotBoundException ex) {
21.                 Logger.getLogger(ClienteSensorEntradaRMI.class.getName()).log(Level.SEVERE, null, ex);
22.             }
23.
24.             try {
25.                 objSensorEntrada.Suscribirse(objClienteSensorEntrada);
26.             } catch (RemoteException ex) {
27.                 Logger.getLogger(ClienteSensorEntradaRMI.class.getName()).log(Level.SEVERE, null, ex);
28.             }
29.         }
30.     }
```