Java y multiprogramación sobre red

Sistemas Distribuidos – ITInformática (UVA)

César Llamas Bello – © 2003

Índice

Multienhebrado en Java *hilos*

- El hilo es la unidad de ejecución del S.O.
- Multienhebrado (procesos ligeros)
  - Todos los hilos acceden a los mismos recursos
  - Cada hilo tiene su propia pila, PC, etc.
- Los hilos pueden ejecutarse en diferentes procesadores, o ser simulados.

Multienhebrado en Java *soporte Java*

- Versiones:
  - Green Threads: la maquina virtual soporta los hilos en nombre del S.O.
  - Native Threads: se aprovecha el multienhebrado del S.O.
- Java soporta multienhebrado ...
  - ... por construcción, y
  - todos los programas Java son Multienhebrados (Reference Handler, Finalizar, Signal Dispatcher, AWT-Motif, ...)

Multienhebrado en Java *soporte Java*

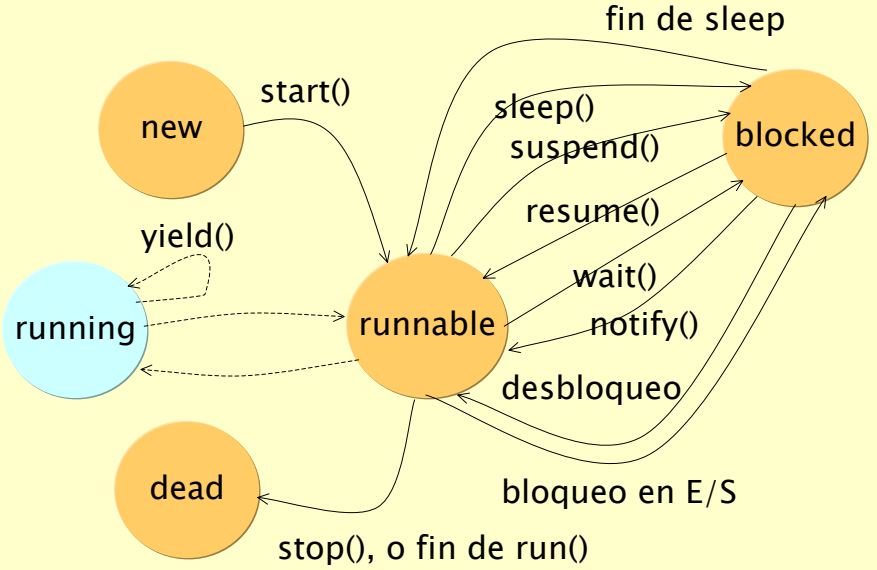
- Elementos Java para el multienhebrado:
  - Se crean hilos:
    - extendiendo la clase Thread, o
    - implementando la interfaz Runnable.
  - Se pueden bloquear recursos mediante el modificador synchronized.
  - Se pueden sincronizar productores y consumidores con wait(), notify() y notifyAll().

Multienhebrado *ejemplo*

- Un reloj en trasfondo:

```
public class TimePrinter implements Runnable {
  public void run() {
    while (true) {
      System.out.println(new Date());
      try {   Thread.sleep(1000);  } // milis
      } catch (InterruptedException x) { }
    }
  }
  public static void main(String[] args) {
    final Runnable tarea = new TimePrinter();
    new Thread(task, “Hilo de TimePrinter”).start();
    ... // resto del programa
  }
}
```

Multienhebrado *estados de un hilo*



Multienhebrado *prioridades*

- Cada hilo tiene una prioridad
  - MIN_PRIORITY ... MAX_PRIORITY
  - Heredada del hilo padre
  - Existe getPriority() y setPriority()
- Los hilos java son desalojables (apropiables, *preemtibles*)
- La planificación es de turno rotatorio (*round-robin*) aunque se pueden poner hilos no desalojables.

Sincronización

- La sincronización para el acceso a regiones críticas se realiza mediante synchronized, a modo de monitores
  - synchronized (referencia a un objeto) { ... }
  - ... synchronized método_sincronizado(...) ... { ... }
- Es posible utilizar puntos de sincronización al estilo wait/notify.

Sincronización *ejemplo productor-consumidor*

```
import java.util.*;

class ColaStrings {
  private ArrayList lista = new ArrayList();

  public synchronized void push(final String p) {
    lista.add(p);
    this.notify(); // hace saber que ha llegado un String
  }
  public synchronized String pop() {
    while (lista.size() == 0) try {
      this.wait(); // espera la llegada de un String
    } catch (final InterruptedException e) { }
    return (String) lista.remove(0);
  }
}
```

Sincronización *ejemplo productor-consumidor*

```
import java.io.*;

class Productor implements Runnable {
  private final ColaStrings cola ;
  public Productor(ColaStrings c) {     this.cola = c;     }
  public void run() {
    final Reader         r1       = new InputStreamReader(System.in);
    final BufferedReader teclado  = new BufferedReader(r1);
    String linea;
    try {
      while ((linea = teclado.readLine()) != null) { // lee teclado
        cola.push(linea);
      }
    } catch (final IOException x) {  x.printStackTrace(System.err);  }
  }
}
```


Sincronización *ejemplo productor-consumidor*

```
class Consumidor implements Runnable {
  private final ColaStrings cola ;
  public Consumidor(ColaStrings c) {
    this.cola = c;
  }
  public void run() {
    String linea;

    while (true) {
      linea = cola.pop();
      System.out.println(">> " + linea); // imprime
    }
  }
```

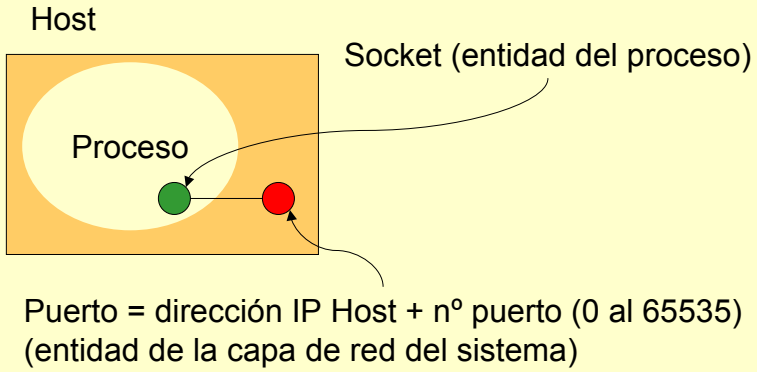
Sincronización *ejemplo productor-consumidor*

```
class LanzaPC {
   public static void main(String[] args) {
      ColaStrings cola = new ColaStrings() ;
      Runnable genera = new Productor(cola) ;
      Runnable come   = new Consumidor(cola) ;

      new Thread(genera).start();
      new Thread(come).start();
   }
}
```


Transporte en Java

- Un poco de Procesos, puertos y sockets
  - El mecanismo de comunicación sobre IP deriva de UNIX y de BSD 4.3.



- Dos versiones de IP
  - IPv4 (32 bits)
  - IPv6 (128 bits) en Java 1.4 se da un API extendido.
- DirecciónIP= entero 32 bits
  - En notación DDN, (ej: 157.88.125.12)
- Nombres host= nombres de dominio
  - Ej.: www.infor.uva.es

- java.net.InetAddress (métodos factoría)
  - Subclases: Inet4Address, Inet6Address

  - static InetAddress getByName(String host)
  - static InetAddress[] getAllByName(String host)
  - static InetAddress getLocalHost()

- Ejemplo:

```
import java.io.*;
import java.net.*;
public class IPAddrTest {
  public static void main(final String[ ] args) {
    InetAddress ip=InetAddress.getByName(www.lab.fi.uva.es);
    System.out.println("IpAddress:" + ip.toString());
    String hostname = ip.getHostName();
    System.out.println("Hostname:" + hostname);
  }
}
```


- Sobre el protocolo de red se implementa:
  - UDP (User Datagram Protocol)
    - No fiable
    - Sin conexión
    - DatagramSocket, DatagramPacket, MulticastSocket
  - TCP (Transmission Control Protocol)
    - Fiable
    - Orientado a conexión
    - Socket, ServerSocket

Transporte en Java *TCP/IP*

- Se diferencia entre el proceso iniciador y el proceso con espera pasiva
- Iniciador:
  1. Crea socket
  2. Enlaza el socket a un puerto
  3. Conecta con el socket receptor (si es posible)
  4. Envía y recibe a través de streams.
  5. Se cierran los streams y el socket

Transporte en Java *TCP/IP*

- Proceso en espera (secundador)
  1. Crea **socket de servicio**
  2. Enlaza el socket a un puerto
  3. Se sitúa en espera (accept)
  4. Al aceptar una conexión crea un socket normal sobre otro puerto
     - Envía y recibe a través de streams
     - Si creamos un hilo nuevo para esta comunicación, podemos volver a 3 con el hilo principal.
     - Se cierran los streams y el socket
  5. Se cierra el socket de servicio.

Transporte en Java — *ejemplo-Cliente*

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

public class Cliente {
  public static void main(String[] args) throws IOException {
    String host = args[0];
    String linea;
    Socket sock = new Socket(host, Servidor.PUERTO);

    final Reader         r1   = new InputStreamReader(System.in);
    final BufferedReader teclado  = new BufferedReader(r1);
    OutputStream out = sock.getOutputStream();
    PrintStream outred = new PrintStream(out);
    InputStream  in  = sock.getInputStream();
    Reader       r2  = new InputStreamReader(in);
    BufferedReader inred   = new BufferedReader(r2);
```


Transporte en Java — *ejemplo-Cliente*

- Continúa ...

```
    while ((linea = teclado.readLine()) != null) { // lee el teclado
      outred.println(linea);   // manda al servidor
      linea = inred.readLine() ; // lee echo del servidor
      System.out.println(">> " + linea); // imprime echo del servidor
    }
    outred.close();
    inred.close();
    sock.close();
  }
}

// Se ejecuta con:        java Cliente host-donde-está-el-servidor
```

Transporte en Java — *ejemplo-Servidor*

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

public class Servidor {
  public static final int PUERTO = 0xCAFE;

  public static void main(String[] args) throws IOException {
    ServerSocket servidor = new ServerSocket(PUERTO) ;

    for (;;) {
      try {
        System.out.println("--------------------");
        Socket        sock = servidor.accept() ;

        InputStream    in = sock.getInputStream(); // entrada socket
        Reader         r1 = new InputStreamReader(in);
        BufferedReader inred = new BufferedReader(r1);
```


Transporte en Java — *ejemplo-Servidor*

- Continúa ...

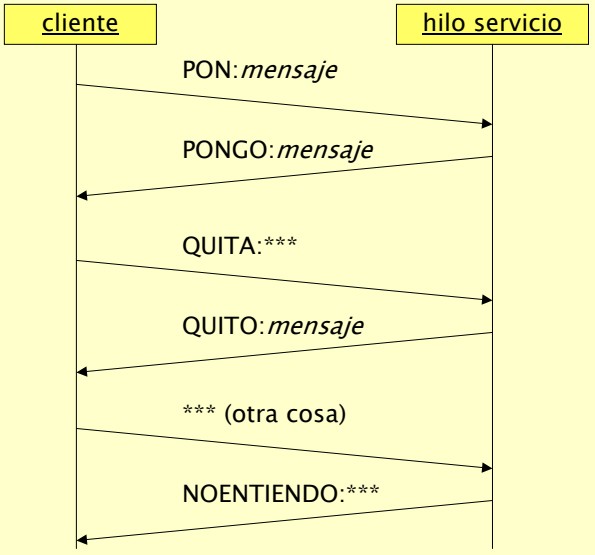
```
        OutputStream   out = sock.getOutputStream(); // salida socket
        PrintStream    outred = new PrintStream(out);
        String linea;

        while ((linea = inred.readLine()) != null) { // lee de la red
          System.out.println(linea);  // echo por la pantalla
          outred.println(linea); // echo al cliente
        }
        inred.close();
        outred.close();
        sock.close();
      } catch (IOException e) {
        e.printStackTrace(System.err);
      }
    } } }
```

Ejemplo conjunto

- Un servidor proporciona acceso a una Cola de Strings (ArrayList de String) para varios procesos:
  - Se requiere sincronización
- en red:
  - Sobre TCP/IP
- El servidor está multienhebrado:
  - Cada cliente debe tener su propio hilo dentro del servidor
- Cada cliente puede depositar o substraer elementos de la cola
  - Se precisa de un protocolo.

Ejemplo conjunto *protocolo de aplicación*



Ejemplo conjunto *Cola de Strings*

```
import java.util.*;

class ColaStrings {
  private ArrayList lista = new ArrayList();

  public synchronized void push(final String p) {
    lista.add(p);
    this.notify(); // hace saber que ha llegado un String
  }
  public synchronized String pop() {
    while (lista.size() == 0) try {
      this.wait(); // espera la llegada de un String
    } catch (final InterruptedException e) { }
    return (String) lista.remove(0);
  }
}
```


Ejemplo conjunto *Cliente*

- El mismo cliente tonto de antes,
- El protocolo tenemos que escribirlo nosotros,
  - No complicamos el cliente,
  - Es más pesado de escribir pero controlamos el protocolo
- Podemos prescindir de nuestro cliente
  - telnet *host* 51966 (*que es el 0xCAFE*)
  - Pero perdemos el control de las interrupciones

Ejemplo conjunto — *Servidor*

```
import java.net.*;
import java.io.*;
public class Servidor {
  public static final int PUERTO = 0xCAFE;
  public static void main(String[] args) throws IOException {
    ServerSocket servidor = new ServerSocket(PUERTO) ;
    ColaStrings cola = new ColaStrings();
    for ( ; ; ) {
      try {
        System.out.println("--------------------");
        Socket       sock = servidor.accept() ;
        Runnable sirviente = new Sirviente(cola, sock);
        new Thread(sirviente).start();
      } catch (IOException e) { e.printStackTrace(System.err);  }
      }
    }  }  }
```


Ejemplo conjunto — *Sirviente*

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

class Sirviente implements Runnable {
  private final ColaStrings    cola ;
  private final Socket         sock;
  private final BufferedReader inred;
  private final PrintStream    outred;

  public Sirviente(ColaStrings c, Socket s) throws IOException {
    this.cola   = c;
    InputStream   in = s.getInputStream(); // entrada socket
      Reader      r1 = new InputStreamReader(in);
      this.inred  = new BufferedReader(r1);
    OutputStream out = s.getOutputStream(); // salida socket
      this.outred = new PrintStream(out);
    this.sock   = s;
  }
```

Ejemplo conjunto *Sirviente*

```
public void run() {
    String  linea, sl;

    try {
        Cuerpo principal
        del hilo
    } catch (final IOException x) {
      x.printStackTrace(System.err);
    }
  }
}
```


Ejemplo conjunto *Sirviente*

```
while ((linea = inred.readLine()) != null) {  // lee de la red
    if (linea.indexOf("PON:") == 0) { // está al principio
        sl = linea.substring(4);
        cola.push(sl);
        outred.println("PONGO:" + sl);
    } else if (linea.indexOf("QUITA:") == 0) {
        linea = cola.pop();
        outred.println("QUITO:"+linea);
    } else {
        outred.println("NOENTIENDO:"+linea);
    }
}
inred.close();
outred.close();
sock.close();
```