Tema 2

Comunicación y sincronización entre procesos en Java

Sistemas Distribuidos

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

Contenido

- Repaso de los conceptos de proceso y *threads*
- Concurrencia
- Mecanismos de comunicación
- Mecanismos de sincronización
- Llamadas al sistema POSIX
  - Para comunicación y sincronización de procesos
  - Gestión de threads
- Threads en C
- Threads en Python
- Threads en Java

Threads en Java

- En Java hay dos formas de crear un proceso ligero:
  - Declarar una clase como subclase de la clase `Thread` y sobrescribir el método `run`. Cuando se inicia la instrancia de la subclase el código del método run se ejecuta de concurrentemente con los hilos ya en ejecución.
  - Declara una clase que implementa la interfaz `Runnable` y sobrescribir el método `run` de dicha interfaz.
- Diferencias entre la utilizar `Thread` o `Runnable`
  - Con Thread a cada hilo se le asocia un objeto distinto. Requiere más memoria
  - Con Runnable todos los hilos comparten los mismos objetos de la clase

Creación de threads en Java extendiendo la clase thread

```
class MyThread extends Thread {
        private String message;
        public MyThread(String m) {message = m;}

        public void run() {
                for (int j=0; j < 100; j++)
                        System.out.println(message);
        }
}

public static void main(String[] args) {
        MyThread t1,t2;
        t1=new MyThread("thread  1");
        t2=new MyThread("thread  2");
        t1.start();
        t2.start();
}
```

Métodos de la clase Thread

- start()
- stop()
- suspend()
- resume()
- setPriority(int)
- getPriority()
- getName()

Creación de threads en Java

```
class MyThread extends Thread {
        private String message;
        public MyThread(String m) {message = m;}
        public void run() {
                System.out.println(message);
                 //Duerme 4 segundos
                 Thread.sleep(4000);
        }
}

public static void main(String[] args) {
        MyThread t1,t2;
        t1=new MyThread("thread  1");
        t1.start();
        t1.join();
}
```

Creación de threads en Java implementando la interfaz Runnable

```
class MyThread implements Runnable {
        private String message;
        public MyThread(String m) {message = m;}

        public void run() {
                for (int j=0; j < 100; j++)
                        System.out.println(message);
        }
}

public static void main(String[] args) {
        MyThread t1,t2;
        t1=new MyThread("thread  1");
        t2=new MyThread("thread  2");
        t1.start();
        t2.start();
}
```

Synchonized Blocks
Exclusión mutua

```
synchronized (object){
        // Lock establecido
... }
// Lock liberado


synchronized void f() {
        /* cuerpo*/
}
```

Es equivalente a:

```
void f() {
        synchronized(this) {
                /* cuerpo*/
        }
}
```

Mientras un thread esté ejecutando un método sincronizado, ningún otro thread podrá ejecutar el mismo bllque

Espera condicional

```
synchronized void EseprarCondicion(){
        while (!condicion)
                try {
                        wait();
                }
                catch(InterrupedException e) {
                        // excepción
                }
                // código a ejecutar si condicion es cierta
}
synchronized void Despertar(){
        condicion = true;
        notify();
}
```

Métodos

- `wait()`
  - El thread actual se bloquea hasta que otro le envíe una notificación al mismo objeto
- `notify()`
  - Activa un thread que está vloqueado en el objeto
- `notifyAll()`
  - Activa a todos los threads, todos los threads compiten, solo uno obtiene el bloqueo

Productor consumidor en Java

```
class Buffer {
  private int[] Datos;
  private int tamano, ocupados, sigEnt, sigSal;

  public Buffer (int tam) {
        datos = new int[tam];
        tamano = tam;
        ocupados = 0;
        sigEnt = 0;
        sigSal =0;
 }

 public synchronized void put (int x) { . . . }

 public synchronized void get (int x) { . . . }
```

Productor consumidor en Java

```
public synchronized void put (int x) {
   try{
      while (ocupados == tamano)
            wait();
      datos[sigEnt] = x;
      sigEnt = (sigEnt+1) % tamano;
      ocupados++;
      notify();
   }
   catch (InterrupedException e) {}
}
```

Productor consumidor en Java

```
public synchronized void get () {
   try{
      while (ocupados == 0)
           wait();
      x=datos[sigSal];
      sigSal=(sigSal+1) % tamano;
      ocupados--;
      notify();
   }
   catch (InterrupedException e) {}
   return x;
}
```

Productor consumidor en Java

```
class Productor extends Thread {
   private Buffer buf;

   public Productor (Buffer b) {
        buf = b;
   }

   public void run() {
      int dato =0;
      while (true){
         buf.put(dato);
         dato++;
      }
   }
}
```

Productor consumidor en Java

```
class Consumidor extends Thread {
   private Buffer buf;

   public Consumidor(Buffer b) {
        buf = b;
   }

   public void run() {
      int dato;
      while (true){
         dato = buf.get();
         System.out.pritln(dato);
      }
   }
}
```

Productor consumidor en Java

```
class ProducorConsumidor{
    public static void main(String [] args{

        Buffer buf;
        buf = new Buffer(200);

        Productor  p = new Productor(buf);
        Consumidor c = new Consumidor(buf);

        p.start();
        c.start();
    }
}
```

Ejercicio
¿Es correcto el siguiente código?

```
class Cuenta{
   long saldo;


   long getSaldo() {
      syncrhonized(saldo) {
         return saldo;
      }
   }


   void retirar(long x){
      if (getSaldo() >= x) {
         syncronized(saldo) {
             saldo = saldo - x;
         }
     }
   }
```

Implementación de semáforos en Java

```
class Semaforo {
  private int count;

   public Semaforo(int c) {
        count = c
   }

   public Semaforo() {
      this(1);
   }
```

Implementación de semáforos en Java

```
synchronized public void acquire {
    count--;
    while (count <0) {
        try {
            wait();
        } catch (Exception e) {System.exit(0);}
    }
  }
synchronized public void release{
    count++;
    if (count <=0)
        notify();
  }
```