Asignación de Prácticas Número 2

Programación Concurrente y de Tiempo Real

Departamento de Ingeniería Informática
Universidad de Cádiz

PCTR, 2022

Contenido I

Objetivos de la Práctica

- Aprender a compartir memoria entre tareas concurrentes
- Conocer el concepto de condición de concurso como ejemplo de condición patológica en un programa concurrente
- Introducir de forma básica el concepto de paralelismo de datos con división manual del dominio

Compartiendo Memoria entre Tareas Concurrentes I

Existen dos formas de compartir información entre tareas concurrentes:

- Utilizando variables de clase (estáticas), que todos los objetos de la clase utilizan de forma compartida
- Una variable es estática si esta cualificada como tal, mediante el cualificador `static`
- Útil para compartir variables de tipos elementales: `int`, `float`, etc.
- Pero esto no sirve si la memoria a compartir involucra a estructuras que necesitan ser más complejas y quizas, incorporar su propia capacidad de procesamiento, o que están predefinidas como clases del lenguaje
- En estos casos, lo que hacemos es compartir objetos de forma que todas las tareas que necesitan compartir un mismo objeto reciben una referencia al mismo

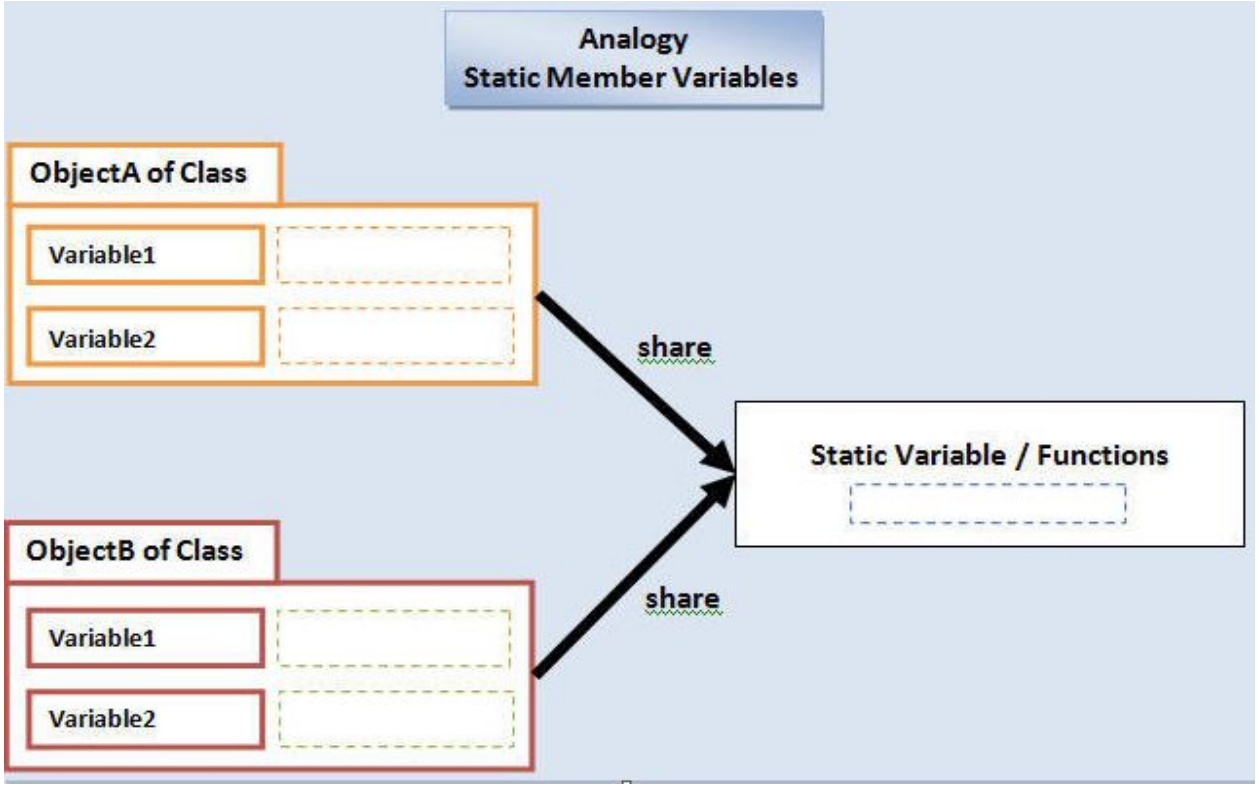
Compartiendo Memoria entre Tareas Concurrentes II

- Como es lógico, cualquiera de ambas técnicas puede dar lugar a problemas de seguridad, vivacidad, etc., si la memoria que compartimos no está protegida frente a accesos concurrentes no seguros
- Buena parte del curso estará dedicada a saber cómo acceder a memoria común de forma segura.

Utilizando Variables Estáticas I

- Es realmente simple; basta cualificar como estáticas a las variables que necesitamos compartir al escribir la clase
- Por diseño, todos los objetos de la clase compartirán esas variables...
- ...incluso si esos objetos son tareas concurrentes modeladas con las técnicas descritas
- Esto debe hacerse bien, o tendremos efectos laterales indeseados
- Para evitar esos efectos, necesitaremos técnicas de control de exclusión mutua aplicadas a las variables estáticas.

Gráficamente...



Condiciones de Concurso I

- Se dan cuando dos o más tareas concurrentes acceden a recursos compartidos entre ellas al mismo tiempo **sin control** de exclusión muta
- Son una fuente inagotable de problemas de seguridad y de vivacidad: incosistencia de datos, interbloqueos, bloques activos, procesos inanes, y en general, todo el espectro de condiciones patológicas habitual en una concurrencia mal controlada
- Hemos de aprender a detectarlas y solucionarlas
- La mejor forma de aprender a detectar condiciones de concurso es escribir unas cuantas, e integrar en nuestros conocimientos la estructura patrón que presentan... para no incurrir en ella.

Resolviendo el Ejercicio 1: Condición de Concurso con Variable Estática I

- Compartiremos la memoria entre hebras utilizadando variables estáticas
- Provocaremos una condición de concurso y una condición patológica (inconsistencia de resultados)
- Ya de paso, dotaremos a nuestra hebras de múltiples comportamientos diferentes, elegidos mediante un parámetro que transferimos al constructor.
- El código que ahora analizamos, es un ejemplo de lo que **no se debe hacer**
- ¿Qué hago ahora? Utilizando como base el código que a continuación analizamos, escriba su propia condición de concurso

Ejemplo de Condición de Concurso con Variable Estática I

```
1  public class Hilo
2    extends Thread
3  {
4      private int tipoHilo;
5      private static int n=0; //variable de clase
6      private int nVueltas;
7
8      public Hilo(int nVueltas, int tipoHilo)
9      {this.nVueltas=nVueltas; this.tipoHilo=tipoHilo;}
10
11     public void run()
12     {
13       switch(tipoHilo){
14         case 0: for(int i=0; i<nVueltas; i++)n++; break;
15         case 1: for(int i=0; i<nVueltas; i++)n--; break;
16       }
17
18     }
19
20   public static void main(String[] args)
21       throws Exception
22   {
```

Ejemplo de Condición de Concurso con Variable Estática II

```
23          Hilo p = new Hilo(10000, 0);
24          Hilo q = new Hilo(10000, 1);
25          p.start();
26          q.start();
27          p.join();
28          q.join();
29          System.out.println(n);
30      }
31
32  }
```

Resolviendo el Ejercicio 2: Condición de Concurso con Objeto Compartido I

- Compartiremos la memoria entre hebras utilizadando un objeto común
- Provocaremos una condición de concurso y una condición patológica (inconsistencia de resultados)
- Veremos cómo transferir la referencias al objeto compartido a todas las tareas mediante el constructor
- Mostramos una solución con herencia de `Thread`... que deberéis adaptar a una implementación de `Runnable`
- El código que ahora analizamos, es un ejemplo de lo que **no se debe hacer**
- ¿Qué hago ahora? Utilizando como base el código que a continuación analizamos, escriba su propia condición de concurso

Resolviendo el Ejercicio 2: Escribimos la Clase que «Compartiremos» I

```
1  public class Critica
2  {
3      private int n=0;
4
5      public Critica() {}
6      public void inc(){n++;}
7      public void dec(){n--;}
8      public int vDato(){return(n);}
9
10
11 }
```

Resolviendo el Ejercicio 2: Escribimos la Condición de Concurso I

```
1  public class hConcu
2    extends Thread
3  {   private Critica c;
4
5      public hConcu(Critica c)
6      {this.c=c;}
7
8      public void run()
9      {
10       for(int i=0; i<10000; i++)c.inc();
11     }
12     public static void main(String[] args)
13       throws Exception
14     {
15       Critica p  = new Critica(); //referencia a objeto comun...
16       hConcu  h1 = new hConcu(p); //ambos hilos comparten el
             acceso a p
17       hConcu  h2 = new hConcu(p); // a traves de la referencia
18       h1.start(); h2.start();
19       h1.join(); h2.join();
```

Resolviendo el Ejercicio 2: Escribimos la Condición de Concurso II

```
20            System.out.println(p.vDato());
21        }
22    }
```

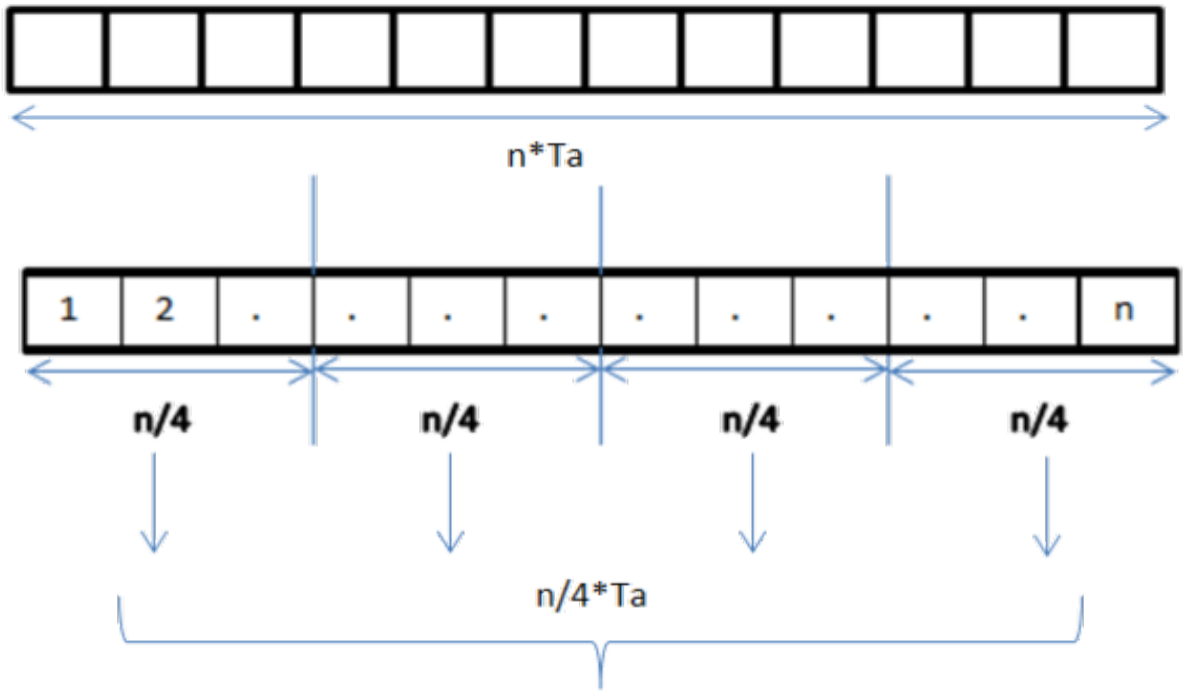
Resolviendo el Ejercicio 3: Introducción al Paralelismo de Datos I

- Se tiene paralelismo de datos cuando múltiples tareas concurrentes se dividen el procesamiento de una nube de datos entre ellas, generalmente en segmentos de datos del mismo tamaño
- Si no existen dependencias de datos y la nube es regular, el paralelismo de datos suele lograr buenos speedups frente a la versión secuencial tradicional
- Es nuestro caso, la nube de datos estará contenida en un array unidimensional...
- ... y las tareas concurrentes/paralelas se dividirán el array en subsegmentos de trabajo iguales.
- Mostramos una solución con herencia de `Thread`... que deberéis adaptar a una implementación con `Runnable`

Resolviendo el Ejercicio 3: Introducción al Paralelismo de Datos II

- El código que ahora analizaremos en un ejemplo de paralelismo con división `manual` del dominio de datos, para un problema trivial: el escalado de un vector
- Es posible también efectuar esa división de forma automática, en función de la plataforma física (número de cores) sobre la que el código se ejecuta. Esto lo hace más transportable.

Gráficamente...



Resolviendo el Ejercicio 3: Producto por Escalar Paralelo (aunque no mucho) I

```
1  import java.util.*;
2  public class XK  extends Thread{
3
4      static double[] data = new double[10];
5      double escalar = 2;
6      int inicio;int fin;
7
8      public XK(int ini, int fin){
9        this.inicio = ini;
10       this.fin    = fin;
11     }
12
13     public void run(){
14       for(int i=inicio; i<=fin; i++)
15         data[i]=data[i]*escalar;
16     }
17
18     public static void main(String[] args)
19       throws Exception{
20       for(int i=0; i<data.length; i++)
```

Resolviendo el Ejercicio 3: Producto por Escalar Paralelo (aunque no mucho) II

```
21            data[i]=Math.random();
22          System.out.println(Arrays.toString(data));
23          XK h1 = new XK(0, 4);
24          XK h2 = new XK(5, 9);
25          h1.start(); h2.start();
26          h1.join(); h2.join();
27          System.out.println(Arrays.toString(data));
28        }
29    }
```

Resolviendo el Ejercicio 3: Cómo Ver el Pico (o la meseta) en la CPU I

- Todos los sistemas operativos tienen una herramienta de línea de comandos o gráfica para monitorizar el rendimiento
- Suelen ilustrar un histórico del % de utilización para cada *core* lógico que el sistema operativo observa
- En Windows, basta pulsar Ctrl-Alt-Supr, escoger «Administrador de Tareas», seleccionar la pestaña de «Rendimiento» y finalmente activar el «Monitor de Recursos».
- En Linux, buscando entre las aplicaciones encontraréis una denominada «Monitor del Sistema»
- Hay dos .pdf en el campus virtual que ilustran cómo hacer esto.

Resolviendo el Ejercicio 3: Cómo Ver el Pico (o la meseta) en la CPU II

- Se trata de ir aumentando el tamaño del array, y el número de hebras, y tratar de visualizar cuánto logramos hacer trabajar a los *cores* con las versiones paralelas. No esperéis grandes cosas, aún aumentado mucho el tamaño del array. El producto es algo tan rápido que introducir paralelismo puede no lograr grandes mejoras sobre la versión secuencial.
- Con nubes de datos reticulares, la cosa cambia, y pueden obtiener buenos speedups.
- IMPORTANTE: la técnica de paralelismo de datos por división manual de la memoria común debería esta clara para todo el mundo. Si no es así, preguntad.

Resolviendo Los Ejercicio 4 y 5: Vosotros Mismos ;-) I

- El ejercicio 4 os pide construir una nueva condición de concurso, ahora sobre una cuenta corriente
- ¿Que debéis hacer?
- Escribir un modelo de una cuenta corriente mediante una clase de Java
- Escribir una condición de concurso entre tareas (que simulan a cajeros automáticos) mediante la técnica de compartir objetos
- Verificar que el saldo final es de todo, menos coherente
- Conclusión: Las bases de datos de los bancos tienen sincronizado el acceso a los datos... lo cuál no sorprenderá a nadie
- El ejercicio 5 es aún más simple, y se trata de tener un par de tareas soportadas mediante λ-expresiones que accedan en condición de concurso a una variable compartida (estática).

Resolviendo Los Ejercicio 4 y 5: Vosotros Mismos ;-) II

- ¿Qué debéis hacer?
- Readaptar el código contenido en `tareaLambda.java`, incluyendo una variable estática a la que accedan concurrentemente vuestras dos tareas.