Marquina Meseguer, Alfredo
Programación Concurrente y Distribuida
Grupo 2, Subgrupo 2
Junio 23/24
Martínez España, Raquel

Proyecto Programación Concurrente y Distribuida

NOTA

Buenas,

Dos aclaraciones sobre esta estrega:

La primera es que he realizado la entrega en este formato, usando una plantilla que tengo de otras asignaturas, para comprobar si sería adecuado (sin esta nota claro) para la entrega final.

La segunda es que he realizado el ejercicio 1 entero con las partes, que tras leer la tarea me he dado cuenta de que no se pedía. Le pido disculpas y que, por favor, las ignore.

Índice

1 EJERCICIO 1

1.1 CÓDIGO

1.1.1 PSEUDO CÓDIGO

```
// Variables goblales
arrayLectura: array[110] de entero
arrayEscritura: array[11] de entero

// También se puede calcular el inicio y final dentro del proceso,
// pero para ello tendríamos que asumir que la distancia entre inicio y final
// es de 11, de esta manera funciona para cualquier N donde el primer elemento
// es un número y la distancia entre inicio y final es impar.
// Si fuera par funcionaría pero la última operación se la comería
proccess type proceso (id:entero, inicio:entero, final: entero, semaforo:semaphore)
begin
      total: entero
      // Si es false el elemento actual es una op si es true es un numero
      esNum: booleano  // Sustituible por: (elementoActual-inicio)%2 == 1
      operación: entero
      temp: entero
end;
begin
      // Inicialización variables
      esNum = false;
      operación = 0;
      // Guardamos el valor del primer elemento pues se añade
      // al total de forma diferente al resto.
      total = arrayLectura[inicio];
      // Recorremos las sección del array dada,
      // salvo el primer que
      for i = incio+1 .. final then
```

```
            temp = arrayLectura[i] // guardamos en local por buenas prácticas
            if esNum then
                  // Si es un número tenemos que añadirlo al total.
                  // Recordemos que la variable operación contiene
                  // el identificador de la operación dada por la
                  // posición anterior an num en el arrayLectura.
                  total = calcular(total, temp, operacion);
            else
                  // Guardamos la operación para poder llamarla con calcular
                  operación = temp;
            finif
            // Como las operaciones y los número se alternan nuestra
            // variable de identificación también.
            esNum = !esNum; // Alternar variable booleana
      finfor
      // Escritura en arrayEscritura
      arrayEscritura[id] = total;
      // La sección crítica solo requiere el uso de la salida
      wait(semaforo);
      writeln(id,total); // Imprimir el mensaje
      signal(semaforo);
end;
/*    Función que realiza una operación sobre los números num1 y num2
      según el valor de op
      Si el valor de op es:
            1 Suma
            2 resta
            3 Multiplicación
            otro error*/
entero function calcular(num1: entero, num2: entero, op: entero)
begin
      total:entero;
end;
begin
```

```
      switch temp then
            case 1:
                  total = num1 + num2;
            case 2:
                  total = num1 - num2;
            case 3:
                  total = num1 * num2;
            default:
                  // Lanzar error
      finswitch
      return total
end;
/*    Caculamos la suma de todos los valores guardados en
      el array de escritura.*/
entero function calculoArrayEscritura():
begin
      total:entero
end
begin
      total =arrayEscritura[0]
      for i = 1 .. len(arrayEscritura)-1:
            total += arrayEscritura[i];
      return total;
end;

main()
begin
      procesos: array[1..11] de procesos;
      solución: entero;
      semaforo: semaphore;
end
begin

      writeln("Comienzo de ejecución");
```

```
      // Definimos procesos
      for i = 0 .. len(arrayEscritura)-1 do
            procesos[i] = proceso(i, i*11, (i+1)*11-1, semaforo)

      // lanzar procesos
      cobegin
      for i = 0 .. len(arrayEscritura)-1 do
            procesos[i].launch();
      coend;
      writeln ("Calculando resultado final");
      solución = calculoArrayEscritura();
      writeln("El total es:",solución);
end;
```

1.1.2 CÓDIGO JAVA

Código de Programa principal

```
public class Ejercicio1 {
      public static int totalArrayEscritura(Integer... arrayEscritura) {
            return Arrays.stream(arrayEscritura).reduce(0, Integer::sum);
      }

      public static void main(String[] args) {
            ReentrantLock lock = new ReentrantLock();
            Integer[] arrayLectura = new Integer[110];
            for (int i =0; i<110; i++) {
                  if (i%11%2==1)
                        arrayLectura[i] = (int) (Math.random() * (3)) + 1;
                  else
                        arrayLectura[i] = (int) (Math.random() * (100)) + 1;
            }
            Integer[] arrayEscritura = new Integer[10];
            HiloEj1[] procesos = new HiloEj1[10];

            System.out.println("Comienzo");

            System.out.println("Creando procesos");
            for (int i = 0; i < 10; i++) {
                  procesos[i] = new HiloEj1(lock, i, i * 11,
(i + 1) * 11 - 1, arrayLectura, arrayEscritura);
            }

            System.out.println("Ejecutando procesos");
            Arrays.stream(procesos).forEach(p -> p.run());

            Arrays.stream(procesos).forEach(p -> {
                  try {
```

```
                        p.join();
                  } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                  }
            });
            System.out.println("Fin de ejecución de procesos. Sumando total.");

            int total = totalArrayEscritura(arrayEscritura);
            System.out.println("El total es: " + total + ".");
      }
}
```

Código Hilo:

```
public class HiloEj1 extends Thread{

      // Semáforo binario, cerrjo, de protección de la sección crítica
      private final ReentrantLock cerrojo;
      // Id del proceso dada por el proceso principal
      private final int processId;
      // Índice del primer elemento del array de lectura permitido
      private final int inicioArrayLectura;
      // Índice del último elemento del array de lectura permitido
      private final int finArrayLectura;
      // Referencia al array de lectura, para acceso al mismo
      private final Integer[] arrayLectura;
      // Referencia al array de escritura, para acceso al mismo
      private final Integer[] arrayEscritura;

      /* Constructor setters y getters */

      public void run() {
            int operacion = 0; // Entero de la última operación encontrada
            int temp; // Elemento actual del array de Lectura
            // Indicador de si el elemento actual es una operación o un número
            boolean esNum = false;
            // Variable donde calculamos el total de la operación
            // El primer elemento se lee diferente
            int total = arrayLectura[inicioArrayLectura];

            // Iteramos desde el inicio permitido hasta el final para leer el
            // array de escritura
            for (int i = inicioArrayLectura + 1; i <= finArrayLectura; i++) {
                  temp = arrayLectura[i];

                  if (esNum) // Si es num calculamos
                        total = calcular(total, temp, operacion);
                  else // Si es operación guardamos para la llamada de calcular
                        operacion = temp;
                  // Alternamos valor como se alterna entre núms y ops
                  esNum = !esNum;
            }
            // Escribir en array de escritura total de operaciones
            arrayEscritura[processId] = total;

            //Sección crítica
            cerrojo.lock(); // Bloquear cerrojo
            // Imprimir mensaje
            System.out.println(
```

```
                        "***************\nProceso terminado: " + processId + "\
nDa como resultado: " + total + "\n***************");
            cerrojo.unlock(); // Liberar cerrojo
      }

      /* Calculo de una operación entre num1 y num2. El valor de operación
      determina la operación a realizar: 1 es suma, 2 es resta y 3 es
      multiplicación. Cualquier otro valor es ilegal.
*/
      private int calcular(int num1, int num2, int operacion) {
            int total;

            try {
                  switch (operacion) {
                  case 1:
                        total = num1 + num2;
                        break;
                  case 2:
                        total = num1 - num2;
                        break;

                  case 3:
                        total = num1 * num2;
                        break;
                  default:
                        throw new IllegalArgumentException("Error calculando la
operacion " + num1 + " " + num2 + " " + operacion + " En el proceso " +
processId);
                  }
            } catch (Exception e) {
                  e.printStackTrace();
                  total = 0;
            }
            return total;
      }
}
```

1.2 CUESTIONES

1.2.1 APARTADO A

¿Qué acciones pueden realizar los hilos concurrentemente? Justifica la respuesta.

Los diferentes procesos pueden realizar las acciones concurrentemente siempre que no estén utilizando un recurso global no compartible. Estos recursos pueden ser tanto software (una variable) como hardware (pantalla), los problemas suelen surgir cuando los procesos escriben en ellos. En este caso, en cada proceso podemos identificar tres recursos que utilizan todos los procesos, el array de lectura, el array de escritura y la pantalla.

El array de lectura (si lo consideramos un solo recurso) cumple la condición de Bernstein ya que ninguno de los procesos, salvo el método main donde no hay concurrencia, edita dicho array, por lo que no hay que preocuparse de su uso.

El array de escritura en un principio parece no cumplir dicha condición pues todos los procesos escriben en él. Sin embargo, si miramos fijamente al comportamiento de los hilos, estos tiene una sola posición donde pueden escribir, sin compartirla con ningún otro proceso. Esto provoca que este recurso no sea, de hecho, compartido, por lo que no hace falta defenderlo. De la misma manera se podría analizar el array de lectura, sin embargo, como nunca se llega a escribir en este no es necesario.

Finalmente, tenemos la pantalla, que es accedida mediante la llamada a sistema `System.out.println`. Este sí que se trata de un recurso no compartible, pues si bien cada proceso es diferente, todos comparten la misma consola de salida estándar.

Con esto llegamos a la conclusión de que todas las acciones se pueden realizar concurrentemente salvo las impresiones por pantalla que tienen que ser protegidas.

1.2.2 APARTADO B

Las impresiones que hacen los hilos, ¿son consecutivas o están desordenadas con las de los demás hilos? ¿Cuál de las opciones consideras que es la correcta? Justifica la respuesta.

Las impresiones realizadas por los diferentes hilos se realizan de forma desordenada según cada proceso llegue a la sección crítica. Esto es porque los procesos se ejecutan en un orden parcial, es decir, que los procesos se pueden ejecutar en un orden diferente al llamado. Sin embargo, es probable que la primera llamada sí que sigan dicho orden y se vallan cambiando mientras las instrucciones de los procesos se vayan entremezclando.

En este ejercicio, parece que los procesos se están ejecutando de forma síncrona, en un orden total, porque el primer hilos en terminar es el primero y sigue el orden hasta el final. No obstante, es probable que esto se deba a que la cantidad de código a ejecutar es muy pequeña y es igual de larga en todos ellos. Es posible, que para poder observar la diferencia sea necesario un problema de mayor magnitud y/o que la ejecución varíe según la entrada. Si se quisiera observar en este problema, se podría pausar la función del proceso una cantidad aleatoria de segundos (por ejemplo, entre 0 y 3).

1.2.3 APARTADO C

Si no usaras ningún mecanismo para sincronización, ¿cómo podría ser la salida en pantalla del programa anterior?

Desconozco exactamente cómo se comunica las función *System.out.println* con la salida estándar, pero parece utilizar (por las pruebas que he realizado) un buffer de strings y consideramos que cada llamada inserta el mensaje correspondiente en la cola.

Dado este caso, mi programa debería seguir ejecutando correctamente ya que solo realizo una llamada por proceso. Sin embargo, si hubiese realizado varias llamadas, una

por cada línea que imprime el proceso, se podrían haber mezclado las líneas de distintos procesos creado un mensaje difícil de leer o resultar imposible asociar el valor del programa con su id.

1.3 Recursos No Compartibles Y Sincronización

Como se ha mencionado en las cuestiones, el único recurso no compatible de este es la pantalla.

En cuanto a sincronización no ha necesitado ninguna sincronización de la concurrencia y el único semáforo utilizado ha sido el cerrojo que tienen todos los procesos. El único orden que debe seguir la ejecución, es:

1. Determinar valor array de escritura si no viene dado
2. Inicializar procesos
3. Ejecutar procesos y esperar que terminen
4. Calcular suma total del array de escritura, editado por los procesos.

Esta “sincronización” ocurren debido a que estas acciones se realizan en el hilo principal que es síncrono.