**Entregable #7.1 [ejercicios] [semana 9 Abr 2018]**

* **Objetivos específicos** que persigue la realización del trabajo.
  + - Practicar con el entorno de Scala sobre eclipse
    - Utilizar algunas de las características del lenguaje Scala diferentes de Java
    - Practicar con el modelo de threads de Scala y su sincronización básica (synchronized)
    - Observar la atomicidad y las condiciones de carrera en programas concurrentes; exclusión mutua
* **Competencias**: CC14, CT7
* **Resultados de aprendizaje**: RA312, RA317,

**Apellidos y nombre**

|  |
| --- |
| Altamirano Cargua Fabricio |
| Gutierrez Ospina Anderson |
|  |
|  |

1. **Ejecute el siguiente programa**

**import** de.\_

**object** Ejemplo1 **extends** App {

**var** uidCount = 0L

**def** getUniqueId() = {

**val** freshUid = uidCount + 1

uidCount = freshUid

freshUid

}

**def** printUniqueIds(n: Int): Unit = {

**val** uids = **for** (i<- 0 until n) **yield** getUniqueId()

log(s"Generated uids: $uids")

}

**val** t = thread { printUniqueIds(5) }

printUniqueIds(5)

t.join()

}

7.1.1 ¿Cuál es el resultado?

**Resultado:**



La L en la declaración uidCount = 0L indica que se trata de un integer literal Long (sección 5.1, sección 5.2). La expresión **for** (i<- 0 until n) **yield** getUniqueId() es un iterador (sección Step 6 del capítulo2, sección 7.3). Esta expresión invoca n (desde 0 a n-1) veces a la función getUniqueId()y almacena el resultado en un intervalo que se asigna a uids.

Como puede observarse los identificadores generados por ambos threads no son únicos (se pretendía que fuesen únicos). Quizás en otra ejecución sí sean distintos. Estamos frente a una **condición de carrera**. Las sentencia **val** freshUid = uidCount + 1 y la sentencia uidCount = freshUid no se ejecutan de manera **atómica**. Para garantizar la atomicidad, Scala define el método synchronized que se puede invocar sobre cualquier objeto. El programa resultante se ilustra a continuación:

**import** de.\_

**object** Ejemplo2 **extends** App {

**var** uidCount = 0L

**def** getUniqueId() = **this**.synchronized {

**val** freshUid = uidCount + 1

uidCount = freshUid

freshUid

}

**def** printUniqueIds(n: Int): Unit = {

**val** uids = **for** (i<- 0 until n) **yield** getUniqueId()

log(s"Generated uids: $uids")

}

**val** t = thread { printUniqueIds(5) }

printUniqueIds(5)

t.join()

}

**2. Ejecute este programa varias veces y explique el resultado:**

7.1.2 ¿Cuál es el resultado?

**Resultado:**

La función getUniqueId()y almacena el resultado en un intervalo que se asigna a uidCount, con el método synchronized nos asegura que al ejecutarse los métodos (printUniqueIds) en paralelo, accedan a la variable uidCount en exclusión mutua,de esta manera se asegura que el resultado sean uids diferentes.



Conseguir exclusión mutua (con synchronized) no es gratuito: se paga en términos de eficiencia. No obstante, siempre que haya memoria compartida hay que asegurar la exclusión mutua en el acceso a la misma. Incluso en situaciones en las que no parece necesario, nos podemos llevar una sorpresa. Estudie el siguiente programa:

**import** de.\_

**object** ThreadSharedStateAccessReordering **extends** App {

**for** ( i <- 0 until 100000) {

**var** a = **false**

**var** b = **false**

**var** x = - 1

**var** y = - 1

**val** t1 = thread {

a = **true**

y = **if** (b) 0 **else** 1

}

**val** t2 = thread {

b = **true**

x = **if** (a) 0 **else** 1

}

t1. join( )

t2. join( )

assert( ! ( x == 1 && y == 1) , s" x = $x, y = $y" )

}

log ("fin")

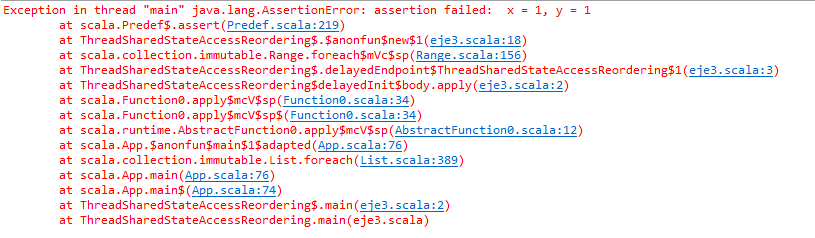
}

Dos threads, t1 y t2, acceden a dos variables booleanas, a y b, y a un par de variables integer, x e y. El valor final de x e y depende de las velocidades relativas de los threads; hay, por tanto, una condición de carrera. Lo que nunca debería obtenerse es que tanto x como y valgan 1 al final. Para tratar ese caso está la sentencia assert (sección 14.1)

1. **Ejecute este programa varias veces y explique el resultado (**Si no sale nunca el string fin prueba a poner la iteración a un valor menor, como 100**):**

7.1.3 ¿Cuál es el resultado?

**Resultado:**



Para evitar esta casuística basta con conseguir la exclusión mutua sobre las variables compartidas.

7.1.4 ¿Qué hay que añadir al programa anterior para conseguir dicha exclusión mutua? Escriba aquí el programa resultante.

Usamos en método synchronized() para así conseguir la exclusión mutua.

