Sistemas Operativos
2021 / 2022

Licenciatura em Engenharia Informática

Lab. 09 – Sincronização de threads em Java

Nesta aula pretende-se que os alunos fiquem com uma noção prática de alguns mecanismos para a sincronização de threads em Java.

Ex. 1 – Sincronização de threads

O seguinte programa define uma classe que implementa um método *count()* para contar os números de 0 a 9. A contagem é executada por duas threads.

```
1  public class Counter {
2
3      static class CounterThread extends Thread {
4          Counter counter;
5
6          public CounterThread(Counter counter) {
7              this.counter = counter;
8          }
9
10         @Override
11         public void run() {
12             counter.count(this.getName());
13         }
14     }
15
16     public void count(String name) {
17         for (int i=0; i<10; i++) {
18             System.out.println(name + ": " + i);
19         }
20     }
21
22     public static void main(String[] args) {
23         Counter counter = new Counter();
24         CounterThread thread0 = new CounterThread(counter);
25         CounterThread thread1 = new CounterThread(counter);
26
27         thread0.start();
28         thread1.start();
29     }
30 }
```

Crie um novo projecto Java no seu IDE e coloque o código anterior num ficheiro de nome *Counter.java*. O *output* deverá ser semelhante ao seguinte:

```
Thread-0: 0
Thread-0: 1
Thread-0: 2
Thread-0: 3
Thread-1: 0
Thread-1: 1
Thread-0: 4
...
```

De uma forma geral, o programa cria um objecto do tipo *Counter* que é passado por parâmetro para as duas threads. Estas, por sua vez, invocam o método *count()*, cuja ordem de contagem interna depende do escalonamento das várias threads.

a) Execute o programa várias vezes e verifique que a contagem dos números não segue um padrão e varia conforme a alternação entre as threads.

b) Pretende-se que cada thread inicie a contagem de forma a que a mesma nunca seja interrompida por outra thread. O que acontece quando acrescenta *synchronized* à assinatura do métod *count()*? Pesquise e comente a solução.

c) Usando o par de comandos *wait()* e *notify()*, altere o método *count* de forma a que a contagem dos números entre as duas threads seja alternada (ex: 0, 0, 1, 1, 2, 2, …). Sugestão: comece por colocar *wait()* a seguir ao *println*.

Ex. 2 – Sincronização de threads usando *wait, notify* e *notifyAll*

O seguinte programa implementa duas threads que estão em ciclo a imprimir os valores colocados no objecto *storage* na função *main*.

```
1  public class Main {
2
3      static class Worker extends Thread {
4          Storage storage;
5
6          Worker(Storage storage) {
7              this.storage = storage;
8          }
9
10         @Override
11         public void run() {
12             while (true) {
13                 System.out.println(getName() + ": " + storage.get());
14             }
15         }
16     }
17
18     static class Storage {
19         int value;
20
21         void set(int value) {
22             this.value = value;
```

```
23          }
25
26          int get() {
27              return this.value;
28          }
29      }
30
31      public static void main(String[] args) throws InterruptedException {
32          Storage storage = new Storage();
33          new Worker(storage).start();
34          new Worker(storage).start();
35
36          for (int i = 0; i < 10; i++) {
37              Thread.sleep(500);
38              storage.set(i);
39          }
40      }
41  }
```

a) Implemente o programa acima mencionado e verifique o output.

b) Modifique a classe *Storage* de modo a que os métodos *get* e *set* estejam acessíveis apenas para uma *thread* de cada vez.

c) Modifique o método *get* da classe *Storage* de forma as *worker* threads sejam obrigadas a aguardar por uma notificação. Essa notificação deverá ser iniciada quando for colocado um valor no objecto *storage* usando o método *set*. Para cada valor (0..9), apenas uma thread deverá imprimir o valor na consola.

d) Modifique o anterior de modo a que todas as threads possam imprimir, pelo menos uma vez, cada um dos valores de 0..9. Sugestão: pesquise pelo método *notifyAll*.

Ex. 3 – Utilização de semáforos em Java

Considere o seguinte programa:

```
1   public class MyThread extends Thread {
2
3       @Override
4       public void run() {
5           doJob(this.getName());
6       }
7
8       static void doJob(String name) {
9           System.out.println("Start: " + name);
10          sleep(2000);
11          System.out.println("End: " + name);
12      }
13
14      public static void main(String[] args) {
15          for (int i = 0; i < 5; i++) {
16              new MyThread().start();
17          }
18      }
19  }
```

a) Implemente o programa e verifique o seu funcionamento.

b) Modifique o programa de modo a que o método *doJob(String)* seja acessível apenas a uma thread de cada vez, usando a *keyword synchronized*.

c) Remova a *keyword synchronized* e modifique o programa para que use semáforos (*java.util.concurrent.Semaphore)* de modo a limitar a implementação do método *doJob(String)* a uma thread de cada vez. O que acontece se inicializar o semáforo com números maiores que 1?

Ex. 4 – Ping-Pong

Edite, compile e execute um programa que recorrendo a duas threads implemente o Ping-Pong. Uma thread deverá escrever "Ping" na consola enquanto a outra deverá escrever "Pong", isto de forma alternada. O "Ping" deverá iniciar primeiro.

Sugestão: considere usar duas classes (de nome *Ping* e *Pong*) e dois semáforos (com nomes do tipo *doPing* e *doPong*) para controlar a ordem de execução das threads.

Ex. 5 – Escritores

Considere um programa onde 10 escritores (*Writers*) preenchem uma lista de inteiros com valores positivos menores ou iguais a 100. Cada escritor demora um intervalo de tempo aleatório nesse preenchimento (menor ou igual a 3000 ms), e apenas preenche uma posição nessa lista. Após os escritores terminarem a sua execução, essa lista de inteiros é escrita no terminal, dentro do método *Main.*

```
1  import java.util.Random;
2  import java.util.logging.Level;
3  import java.util.logging.Logger;
4
5
6  public class Writer extends Thread {
7      static Storage storage;
8      int index;
9
10     Writer (int index) {
11         this. index = index;
12     }
13
14     @Override
15     public void run() {
16         Random r = new Random();
17         int time = r.nextInt(2000)+1;
18
19         System.out.println("Writer " + index + "# sleep time = " + time);
20
21         try {
22             Thread.sleep(time);
23         } catch (InterruptedException ex) {
23             System.out.println(ex.getMessage());
```

```
24          }
25          storage.set(index, r.nextInt(100)+1);
26      }
27
28      static class Storage {
29          int [] list = new int[10];
30
31          void set(int index, int value) {
32              this.list[index] = value;
33          }
34
35          int [] get() {
36              return this.list;
37          }
38      }
39
40      public static void main(String[] args) throws InterruptedException {
41          Storage storage = new Storage();
42          Writer [] writers = new Worker[10];
43
44          Writer.storage =  storage;
45          long lStartTime = System.currentTimeMillis();
46
47          for (int i = 0; i < 10; i++) {
48              writers [i] = new Writer (i);
49              writers [i].start();
50          }
51
52          long lEndTime = System.currentTimeMillis();
53          int [] localList = storage.get();
54
55          for (int i = 0; i < 10; i++) {
56              System.out.println(i+" - "+localList[i]);
57          }
58          System.out.println("Reading results after "+ (lEndTime-lStartTime)
59                             + " milisecs");
60      }
61  }
```

a) Implemente o programa e verifique o seu funcionamento.

b) Adicione dentro do método *Main* uma instrução *sleep* com o intervalo de tempo que achar desejável, de forma a que **imediatamente após** todos os escritores terminarem a sua escrita, os resultados sejam mostrados no ecrã.

c) De forma que a operação pedida na alínea anterior seja executada com a precisão pedida, adicione as instruções *wait* e *notify* aos métodos da classe *Storage*.

(fim de enunciado)