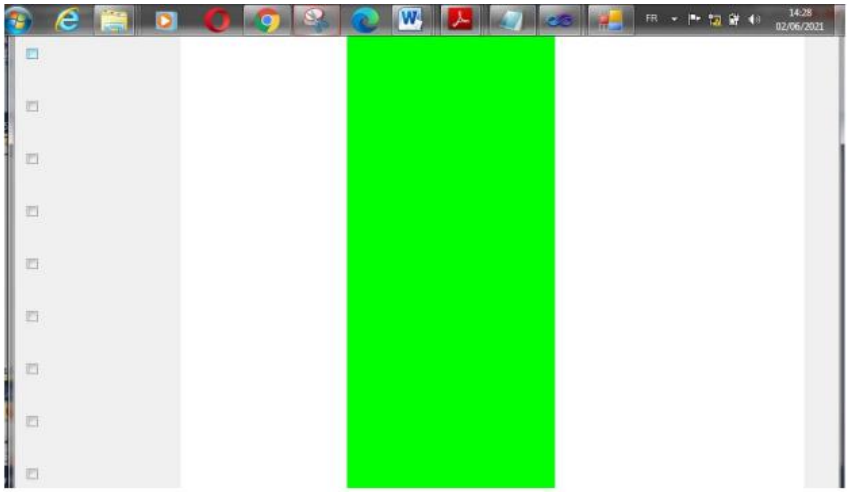
TP 7

Pour la leçon 5

Code source:

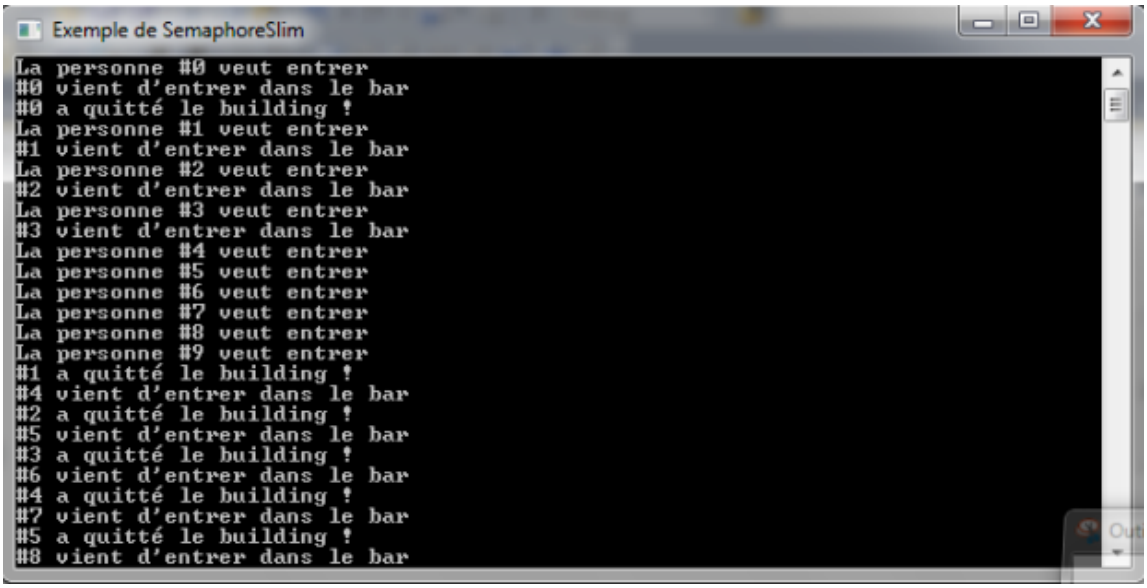
```
Imports System
Imports System.Collections.Generic
Imports System.Text
namespace SynchronizedBalls
public class Forks
 bool[] fork = new bool[5]
// initially false, i.e. not used
// Try to pick up the forks with the designated numbers
public sub Get(int left, int right)
lock (this)
while (fork[left] || fork[right])
Monitor.Wait(this)
//
BallMov bl = new BallMov(pbox)
fork[left] = true
fork[right] = true
Next
End sub
// Lay down the forks with the designated numbers
public sub Put(int left, int right)
lock (this)
fork[left] = false
fork[right] = false
Monitor.PulseAll(this)
End sub
End class
```

Capture



b. Leçon 6

Solution



Code sources

```
public class semaphore

//Déclaration du SemaphoreSlim qui prendra en paramètre le nombre de places disponibles.

static SemaphoreSlim doorman = new SemaphoreSlim(3)

public sub semaph()

Console.Title = "Exemple de SemaphoreSlim"

 //Création des threads.
```

```
      For K As Int = 0 to 9
      new Thread(Entrer).Start(i)
Next
Console.ReadKey()
static sub Entrer(object n)
Console.WriteLine("La personne #{0} veut entrer", n)
//Le doorman attendra qu'il y ait de la place.
doorman.Wait()
Console.WriteLine("#{0} vient d'entrer dans le bar", n)
Thread.Sleep((int)n * 1000)
Console.WriteLine("#{0} a quitté le building !", n)
//Le doorman peut maintenant faire entrer quelqu'un d'autre.
doorman.Release()
public sub rt()
End sub
```

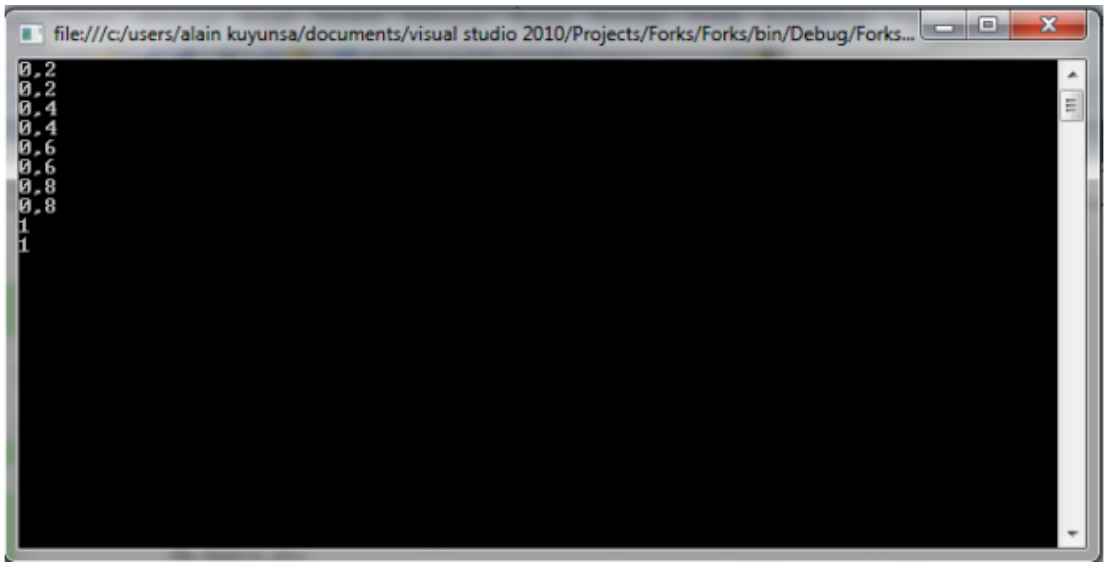
c. La solution « philosophe » n'utilise pas des locks, mais uniquement la classe Monitor

```
Imports System
Imports System.Collections.Generic
Imports System.Text
namespace SynchronizedBalls
public class Forks
bool[] fork = new bool[5]
// initially false, i.e. not used
// Try to pick up the forks with the designated numbers
public sub Get(int left, int right)
lock (this)
while (fork[left] || fork[right]) Monitor.Wait(this)
 //BallMov bl = new BallMov(pbox)
fork[left] = true
fork[right] = true
End sub
// Lay down the forks with the designated numbers
```

```
public sub Put(int left, int right)
 lock (this)
fork[left] = false;
 fork[right] = false
Monitor.PulseAll(this)
 End sub
End class
```

d. La solution « philosophe » doit respecter les règles concernant l'objet synchronization qui ont été énoncées dans les diapositives PowerPoint (diapositive 7).

Solution



Code source

```
public class Forkss
bool[] fork = new bool[5]
// initially false, i.e. not used
public STEPS = 5 As Double
public PercentageTakenLeft As Double
public PercentageTakenRight As Double
// Try to pick up the forks with the designated numbers
public sub Get(int left, int right)
lock (this)
while (fork[left] || fork[right])
```

```
Monitor.Wait(this)

ReturnBothSticks()

Put(2,2)

fork[left] = true

fork[right] = true

 End sub

// Lay down the forks with the designated numbers

public void Put(int left, int right)

 lock (this)

fork[left] = false

TakeBothSticks()

fork[right] = false

ReturnBothSticks()

Monitor.PulseAll(this)

End sub

public sub TakeBothSticks()

For K As Int = 0 to STEPS
PercentageTakenLeft += 1.0 / STEPS

PercentageTakenRight += 1.0 / STEPS

Thread.Sleep(100)

Next

Console.WriteLine(PercentageTakenLeft)

Thread.Sleep(100)

Console.WriteLine(PercentageTakenRight)

End sub

public sub ReturnBothSticks()

For K As Int = 0 to STEPS
Thread.Sleep(100)

PercentageTakenLeft -= 1.0 / STEPS

Thread.Sleep(100);

PercentageTakenRight -= 1.0 / STEPS

Next

End sub

End class

class Program
```

```
static sub Main(string[] args)
Forkss fk = new Forkss()
fk.Put(1,2)
Thread.Sleep(150)
Thread.CurrentThread.Interrupt()
fk.Get(2,2)
Console.ReadKey()
End sub
End class
```