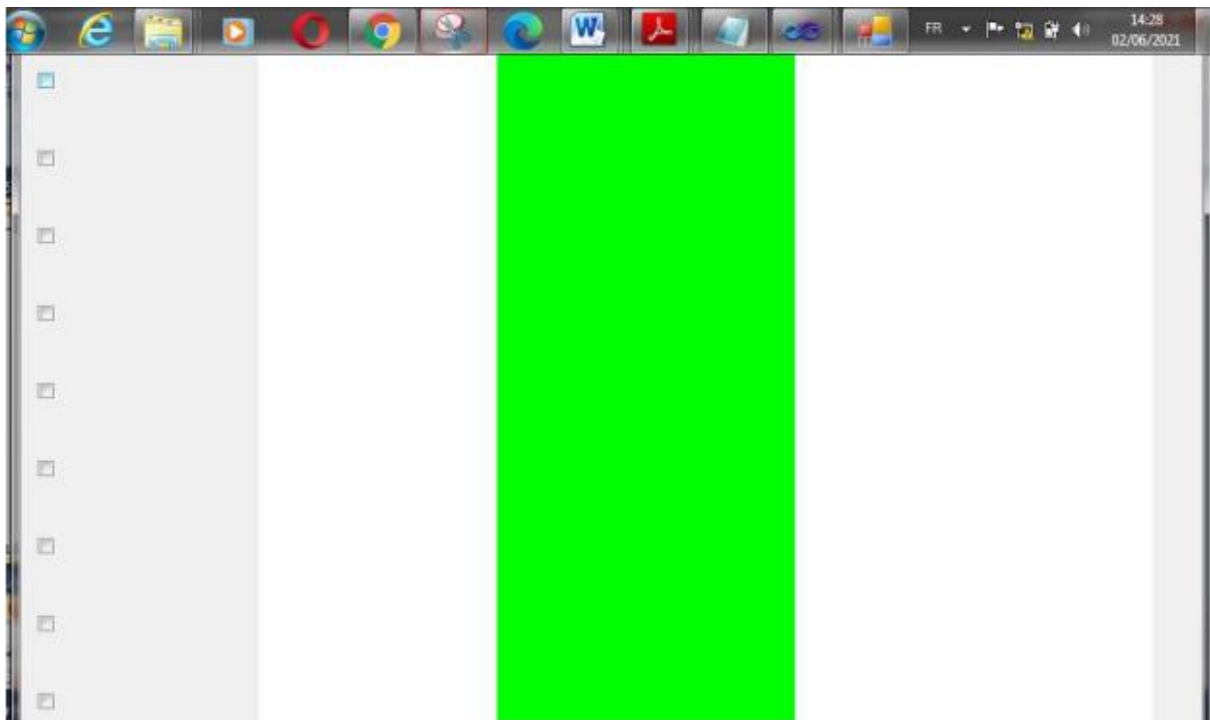


This week we will solve the Readers/Writers problem and the problem of the Dining Philosophers again, but now using a VB Monitor. a. Take the solution for the Readers/Writers problem that you made in lesson 5 or 6 of this course. Change that solution such that the class RW from the slides of this week is used (that class is not completely given in the sheets, so you must complete it first). Clearly, no semaphores should be used and also not the class ReaderWriterLock(Slim). As an experiment, replace the while by if (twice) and check that the result is behaving wrongly.

for lesson 5

solution



code source:

```
Imports System
```

```
Imports System.Collections.Generic
```

```
Imports System.Text
```

```
namespace SynchronizedBalls
```

```
public class process
```

```
    bool[] process = new bool[5]
```

```

public sub Get(int left, int right)
lock (this)
while (process[left] || process[right])
Monitor.Wait(this)
//
BallMov bm = new BallMov(pitbox)
process[left] = true
process[right] = true
Next
End sub

```

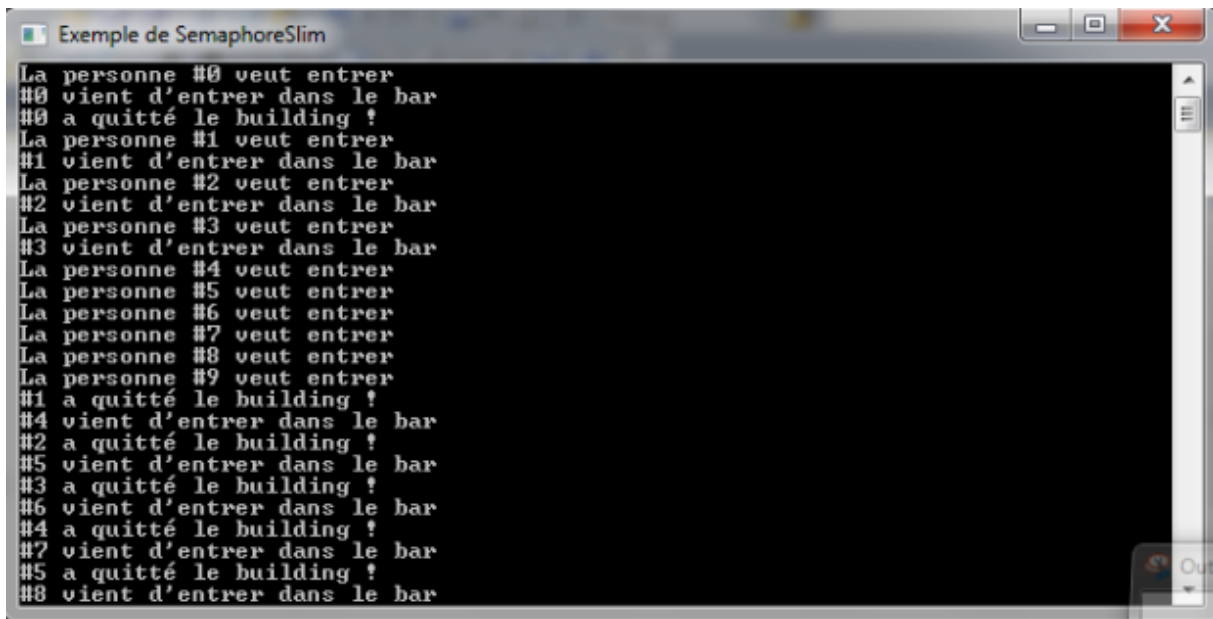
```

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

```

b. Take again the Philosophers project of lesson 6. Solve the deadlock problem again, but now let the philosophers take 2 sticks at the same time and use the C# Monitor class to program this (don't use Semaphores). (If you are unable to do this on your own, take the Forks class from GloogleClassroom and add it to your project. Of course, you have to rename that class to Sticks, because in our story the philosophers use sticks instead of forks. Change the Philosopher class such that it uses the Sticks class.) In order for the drawing to be done right, add the methods TakeBothSticks and ReturnBothSticks to the Philosopher class. These methods can be found in file "new methods.txt". Call TakeBothSticks after the philosopher gets the state Eating, call ReturnBothSticks before the philosopher gets the state Thinking. Don't use the states LeftStickTaken and RightStickTaken anymore.

Solution



```
Exemple de SemaphoreSlim
La personne #0 veut entrer
#0 vient d'entrer dans le bar
#0 a quitté le building !
La personne #1 veut entrer
#1 vient d'entrer dans le bar
La personne #2 veut entrer
#2 vient d'entrer dans le bar
La personne #3 veut entrer
#3 vient d'entrer dans le bar
La personne #4 veut entrer
La personne #5 veut entrer
La personne #6 veut entrer
La personne #7 veut entrer
La personne #8 veut entrer
La personne #9 veut entrer
#1 a quitté le building !
#4 vient d'entrer dans le bar
#2 a quitté le building !
#5 vient d'entrer dans le bar
#3 a quitté le building !
#6 vient d'entrer dans le bar
#4 a quitté le building !
#7 vient d'entrer dans le bar
#5 a quitté le building !
#8 vient d'entrer dans le bar
```

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. Make your ‘Philosopher’ solution such that no lock is used anymore, but only the Monitor class.

```
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 bm = new BallMov(pitbox)
```

```
process[left] = true
```

```
process[right] = true
```

```
End sub
```

```
// Lay down the forks with the designated numbers
```

```
public sub Put(int left, int right)
```

```
lock (this)
```

```
process[left] = false;
```

```
process[right] = false
```

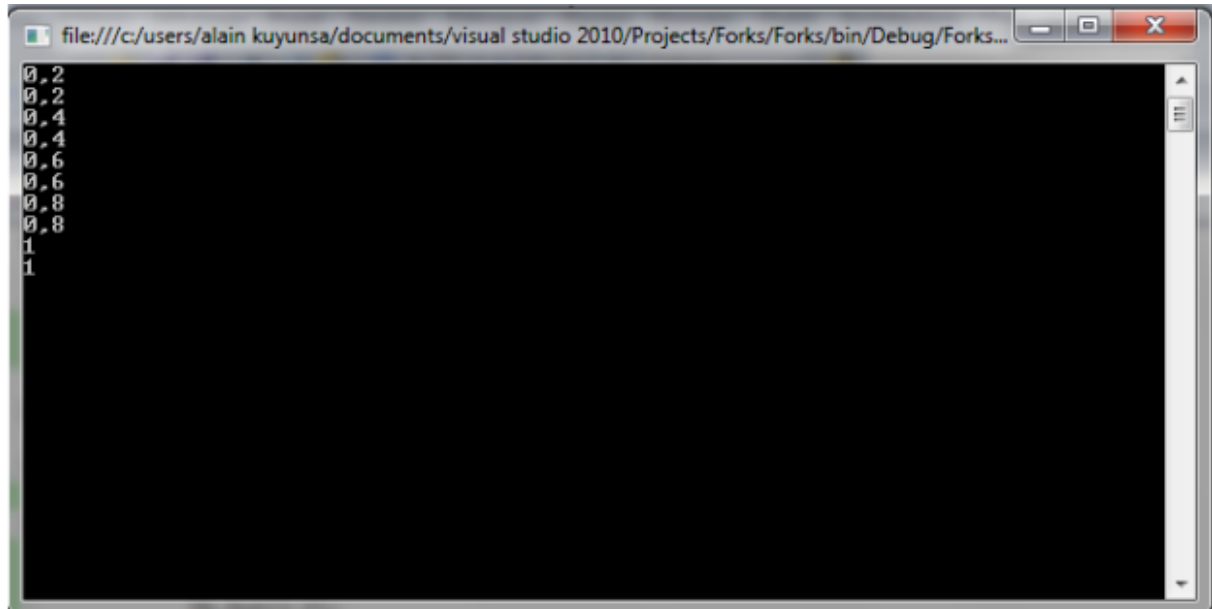
```
Monitor.PulseAll(this)
```

End sub

End class

d. Make sure that your 'Philosopher' solution obeys the rules about the synchronization object that were stated in the PowerPoint slides (slide 7).

Solution



Code source

```
public class process
bool[] process = new bool[5]
// initially false, i.e. not used
public TAILLE= 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 (process[left] || process[right])
Monitor.Wait(this)
ReturnBothSticks()
```

```

Put(2,2)
process[left] = true
process[right] = true
End sub

// Lay down the forks with the designated numbers
public void Put(int left, int right)
    lock (this)
    process[left] = false
    TakeBothSticks()
    process[right] = false
    ReturnBothSticks()
    Monitor.PulseAll(this)
End sub

public sub TakeBothSticks()
    For K As Int = 0 to TAILLE
        PercentageTakenLeft += 1.0 / TAILLE
        PercentageTakenRight += 1.0 / TAILLE
        Thread.Sleep(100)
    Next
    Console.WriteLine(PercentageTakenLeft)
    Thread.Sleep(100)
    Console.WriteLine(PercentageTakenRight)
End sub

public sub ReturnBothSticks()
    For M As Int = 0 to TAILLE
        Thread.Sleep(100)
        PercentageTakenLeft -= 1.0 / TAILLE
        Thread.Sleep(100);

```

PercentageTakenRight -= 1.0 / TAILLE

Next

End sub

End class

class Program

static sub Main(string[] args)

process ps = new process()

ps.Put(1,2)

Thread.Sleep(150)

Thread.CurrentThread.Interrupt()

ps.Get(2,2)

Console.ReadKey()

End sub

End class