CS365 Operating System

Lab 5

Part 1: A single-lane bridge

> A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if both a northbound and a southbound farmer get on the bridge at the same time (Vermont farmers are stubborn and are unable to back up). Using semaphores, design an algorithm that prevents deadlock. Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, and vice versa).

I did part one in Psudocode and can be seen below:

```
Semaphore signal = new Semaphore(1,1);

void enter()
{
    // aquire the lock
    signal.WaitOne();
}

void exit()
{
    // release the lock
    signal.Release();
}
```

Part II –Revise Part I

> Modify your solution to Part I so that it is starvation-free.

The next stage I did in C# and was able to get a running solution. The first thing I did was create a farmer class and a shared ID class so that I could have a referance of what farmer is waiting/crossing/crossed.

```
// Global ID varable storage.
class SharedID
{
    int ID = 0;
    public int GetID()
    {
        ID++;
        return ID;
    }
}
```

```
/// <summary>
/// The farmer accepts a common ID and a common Bridge that it trys to cross while
running.
/// </summary>
class Farmer
{
    private string name;
    public int id { get; set; }
    private Bridge bridge;
    public Farmer(Bridge bridge, SharedID iD)
    {
        this.bridge = bridge;
        this.id = iD.GetID();
    }

    public void Run()
    {
        bridge.crossBridge(this);
    }

    public String GetName()
    {
        return name;
    }

    public void SetName(String name)
    {
        this.name = name;
    }

}
```

The next part that needed to be created was the bridge class which would control the crittical section that would aquire the lock Semephore to access and then release the Semaphore.

```
/// <summary>
/// The bridge controlls the semaphore and the crossing times of the farmers
/// </summary>
class Bridge
{
    private Semaphore signal;
    Random rand = new Random();


    public Bridge()
    {
        signal = new Semaphore(1,1);
    }
    public void crossBridge(Farmer farmer)
    {
        try
```

```
        {
            Console.WriteLine(farmer.GetName() + " [" + farmer.id + "]: "+"is
waiting for signal to cross the bridge.");
            // aquire lock
            signal.WaitOne();
            Console.WriteLine(farmer.GetName() + " [" + farmer.id + "]: " + "is
crossing the bridge.");

            // utilize a random number to set how it takes to cross the bridge
            // sleep for that duration
            Thread.Sleep( rand.Next(0, 1000) * 10 );
        }
        catch (ThreadInterruptedException e)
        {
            Console.WriteLine(e.ToString()); ;
        }
        finally
        {
            Console.WriteLine(farmer.GetName() + " [" + farmer.id + "]: " + "has
crossed the bridge.\n" );
            // release lock
            signal.Release();
        }
    }
}
```

Once these classes had been created I could create the threading to run in the main functions using the following:

```
class SingleLaneBridge
{
    public static void Main()
    {
        Bridge bridge = new Bridge();
        SharedID iD = new SharedID();
        Random rand = new Random();

        // Create a northbound farmers thread
        Thread Northbound = new Thread(new ThreadStart(NorthRun));
        // Create a southbound farmers thread
        Thread Southbound = new Thread(new ThreadStart(SouthRun));

        void NorthRun()
        {

            while (true)
            {
                Farmer farmer = new Farmer(bridge, iD);
                Thread th = new Thread(new ThreadStart(farmer.Run));
                // Sets the farmers name
                farmer.SetName("Northbound Farmer" );
```

```
                th.Start();
                try
                {
                    // how long it will wait before creating another Northbound
farmer
                    Thread.Sleep(rand.Next(0,1000) * 10);
                }
                catch (ThreadInterruptedException e)
                {
                    Console.WriteLine(e.ToString());
                }
            }

        }
        void SouthRun()
        {
            while (true)
            {
                Farmer farmer = new Farmer(bridge, iD);
                Thread th = new Thread(new ThreadStart(farmer.Run));
                // Sets the farmers name
                farmer.SetName("Southbound Farmer");
                th.Start();
                try
                {
                    // how long it will wait before creating another Southbound
farmer
                    Thread.Sleep(rand.Next(0,1000) * 10);
                }
                catch (ThreadInterruptedException e)
                {
                    Console.WriteLine(e.ToString());
                }
            }
        }

    Northbound.Start();
    Southbound.Start();
    }
}
```

Output was as following:

```
Northbound Farmer [1]: is waiting for signal to cross the bridge.
Southbound Farmer [2]: is waiting for signal to cross the bridge.
Northbound Farmer [1]: is crossing the bridge.
Northbound Farmer [3]: is waiting for signal to cross the bridge.
Northbound Farmer [1]: has crossed the bridge.

Southbound Farmer [2]: is crossing the bridge.
Southbound Farmer [2]: has crossed the bridge.
```

```
Northbound Farmer [3]: is crossing the bridge.
Northbound Farmer [3]: has crossed the bridge.

Southbound Farmer [4]: is waiting for signal to cross the bridge.
Southbound Farmer [4]: is crossing the bridge.
Southbound Farmer [4]: has crossed the bridge.

Northbound Farmer [5]: is waiting for signal to cross the bridge.
Northbound Farmer [5]: is crossing the bridge.
Southbound Farmer [6]: is waiting for signal to cross the bridge.
```

From this we can see that while farmer [1] is crossing the bridge southbound farmer [2] and northbound farmer [3] are both waiting for [1] to finish crossing. Once the first one has crossed it allows for the southbound [2] farmer to cross. Thus we have prevented starvation and solved Part 2.