Ikaskuntza Birtual eta Digitalizatuen LHII
CIFP de Aprendizajes Virtuales y Digitalizados

Natalia Carolina Mendez Chapoval
DAM - PSP

Curso: 2021/22

Ud02 TE01
Programación multihilo

EUSKO JAURLARITZA GOBIERNO VASCO
HEZKUNTZA SAILA
Lanbide Heziketako Sailburuordetza
DEPARTAMENTO DE EDUCACIÓN
Viceconsejería de Formación Profesional

ÍNDICE

1. INTRODUCCIÓN

Ejercicio 1

El ayuntamiento de Donostia trabaja con la empresa **Alokatu** de alquiler de bicicletas. Alokatu, necesita una **gestión de Stock**. Las primeras zonas en las que se comenzará a gestionar el alquiler serán la Zona de Amara y la de Gros.

El programa deberá gestionar el Stock de la siguiente forma.

- Existirá una **tarea PRODUCTOR** para simular el trabajo del proveedor. El proveedor proveerá de **200 bicicletas** y éstas se irán cargando a un **almacén principal**.
  - Las bicis se etiquetarán con un número del 1-200.
  - Una vez provistas las 200 bicicletas la tarea finaliza y el almacén principal se marcará como cerrado (completado).
- Se creará una **tarea CONSUMIDOR**, dicha tarea simulará la recogida de bicis del almacén de los usuarios de la **zona de Gros**.
  - Los usuarios irán alquilando bicis del almacén principal.
  - En aquellos casos en los que la etiqueta de bici coincida con un **múltiplo de 3**, se considerará que la bici está lista para su devolución.
    - La devolución será al almacén principal si dicho almacén sigue abierto.
    - La devolución se producirá a un almacén secundario, en caso de que el almacén principal ya esté cerrado.
  - La tarea consumidor finalizará cuando el almacén principal esté vacío y marcado como completado.
- Se creará una **segunda tarea CONSUMIDOR**, esta tarea consumirá bicicletas del almacén central pero desde la **zona de Amara**.
  - Los usuarios irán cogiendo bicis del almacén principal.
  - En aquellos casos en los que la etiqueta de bici coincida con un **múltiplo de 5,** se considera que la bici está lista para su devolución.
    -
    - La devolución será al almacén principal si dicho almacén sigue abierto.
    - La devolución se producirá a un almacén secundario, en caso de que el almacén principal ya esté cerrado.
  - La tarea consumidor finalizará cuando el almacén principal esté vacío y marcado como completado.
- En el programa existirán **dos almacenes**, un almacén PRINCIPAL y otro SECUNDARIO, sus funciones se describen a continuación:
  - **Almacén PRINCIPAL**:
    - Recogerá aquellas bicis del proveedor.
    - Tamaño máximo del almacén, 100 unidades.
    - Cuando el proveedor haya provisto de 200 bicicletas, se marcará como cerrado (completado):
      - El proveedor no podrá proveer más bicis al almacén cuando se marque como completado.
      - Las tareas consumidores no podrán devolver bicis a este almacén cuando se marque como completado.
      - Los consumidores podrán seguir consumiendo bicis de dicho almacén hasta que éste quede vacío.
  - **Almacén SECUNDARIO**.
    - Recogerá aquellas bicis que devuelvan las tareas consumidoras una vez se marque el almacén PRINCIPAL como cerrado.
    - Tamaño máximo del almacén, 100 unidades.
    - No se consumirán bicis por parte de ninguna tarea de dicho almacén.
    - El proveedor no añade ningún elemento a este almacén.
- Todas las acciones realizadas en las tareas tanto de productor como de consumidor deberán aparecer por consola en modo **LOG**. Ejemplo:
  - Número de bicis que se van creando y añadiendo al almacén principal.

- Devolución de bici con número de bici y almacén al que se devuelve.
- Cierre de almacén con el nombre de almacén
- Alquiler de bici, nombre de la zona en la que se consume y número de bici consumida.
- Stock de almacén que queda en el almacén secundario al finalizar todas las tareas.
- Datos relevantes para entender la ejecución del programa.

Para el ejercicio será obligatorio hacer uso de **BlockingCollection** y la clase **Task.**

Ejercicio 2

El objetivo del segundo ejercicio es trabajar los bloqueos con la clase **Thread utilizando Monitores**. Y para ello utilizaremos un ejercicio clásico que lleva por nombre "La cena de los filósofos"

El problema es el siguiente: Cinco filósofos se sientan alrededor de una mesa y pasan su vida comiendo y pensando. Cada filósofo tiene un plato de arroz chino y un palillo a la izquierda de su plato. Cuando un filósofo quiere comer arroz, cogerá los dos palillos de cada lado del plato y comerá. El problema es el siguiente: establecer un ritual (algoritmo) que permita comer a los filósofos. El algoritmo debe satisfacer la exclusión mutua (dos filósofos no pueden emplear el mismo palillo a la vez), además de evitar el interbloqueo.

Descripción o condiciones del programa:

- Existirán **5 filósofos** con su plato asignado.
- Existirán **5 palillos**.
- Cada filósofo, sólo podrá hacer uso de sus palillos (aquellos que estén pegando a su plato).
- Un filósofo puede tener **4 estados**.
  - Estado **COMIENDO**:
    - Está comiendo arroz con los dos palillos que tiene asignados.
    - Cuando pasa un tiempo (1 seg) suelta los palillos.
    - Y pasa a estado INTENTANDO COMER.
  - Estado **HAMBRIENTO**:
    - El filósofo ha conseguido un palillo y está intentando conseguir el segundo.
    - Tras una pequeña espera (1 seg).
      - Si consigue el segundo palillo ==> Pasa a estado COMIENDO.
      - Si no consigue el segundo palillo ==> Pasa a estado INTENTANDO COMER. Y suelta el palillo que había conseguido previamente.
  - Estado **INTENTANDO COMER:**
    - No tiene ningún palillo.
    - Cuando pase un tiempo (1 seg), intenta conseguir uno de los palillos que tiene asignado.
      - Si consigue un palillo ==> Pasa a estado HAMBRIENTO.
      - Si no consigue un palillo ==> Sigue en estado INTENTANDO COMER.
  - Estado **PENSANDO:**
    - Ha conseguido comer con éxito 5 veces.
    - Finaliza el programa.
- No habrá más de 2 filósofos comiendo al mismo tiempo.

Requisitos mínimos del programa:

- Se debe hacer uso de la clase Thread y Monitor. Este ejercicio no está pensado para trabajar con Task.
- Cumplir con las condiciones del programa.
- Mostrar por consola los LOG de estados y acciones que se están realizando.
  - Número de filósofo, estado y acción que está realizando.
  - Palillos asignados al filósofo y si entra a cenar.
  - Finalización de tareas una vez haya comido 5 veces.
  - Más información relevante para entender el programa desarrollado.

- No se produce interbloqueo.

2. DIFERENTES APARTADOS DE LA TAREA

Enlace de acceso a los programas en drive:
https://drive.google.com/file/d/1EWhMlkdiJb39zbF0jm79ZgurmJtLjS2-/view?usp=sharing

Ejercicio 1

PRODUCTOR:

```
// Tarea Productor: agrega 1 bici a almacenPrincipal hasta que bici = 200
Task Productor = Task.Run(() =>
{
    int bici = 1;
    bool AñadirBici = true;

    while (AñadirBici)
    {
        if (!almacenPrincipal.IsCompleted)
        {
            almacenPrincipal.Add(bici);
            {
                Console.WriteLine("El proveedor ha entregado la bici número {0} al almacén principal", bici);
                bici++;
            }
        }
        if (bici == 201) AñadirBici = false;
    }
    // AlmacenPrincipal se cierra al haber recibido las 200 bicis
    almacenPrincipal.CompleteAdding();
    Console.WriteLine("Se cierra el Almacén Principal");
});
```

Provee 200 bicis

Marca el almacen cerrado

CONSUMIDOR GROS:

```
// Tarea Consumidor de Gros
Task Consumidor_Gros = Task.Run(() =>
{
    while (!almacenPrincipal.IsCompleted)
    {
        int bici = -1;

        try
        {
            bici = almacenPrincipal.Take();
        }
        catch (InvalidOperationException)
        {
            Console.WriteLine("Error: no se ha podido acceder al almacén.");
        }
        if (bici != -1)
        {
            Console.WriteLine("Un usuario en la zona de Gros alquila la bici {0} del  almacén principal", bici);
        }
    }
});
```

Alquila bicis del almacén principal mientras no esté cerrado

CONSUMIDOR AMARA:

```
// Tarea Consumidor de Amara
Task Consumidor_Amara = Task.Run(() =>
{
    while (!almacenPrincipal.IsCompleted)
    {
        int bici = -1;

        try
        {
            bici = almacenPrincipal.Take();
        }
        catch (InvalidOperationException)
        {
            Console.WriteLine("Error: no se ha podido acceder al almacén.");
        }
        if (bici != -1)
        {
            Console.WriteLine("Un usuario en la zona de Amara alquila la bici {0} del  almacén principal", bici);
        }

        // Gestion de la devolución cuando el numero de la bici es múltiplo de 5
        if (bici % 5 == 0)
        {
            // se devuelve al almacen principal
            if (!almacenPrincipal.IsAddingCompleted)
            {
                if (!almacenPrincipal.IsCompleted)
                {
                    if (almacenPrincipal.TryAdd(bici, TimeSpan.FromSeconds(1)))
                    {
                        Console.WriteLine("Un usuario en la zona de Amara devuelve la bici {0} al almacén principal", bici);
                    }
                }
            }
            // se devuelve al almacen secundario por estar cerrado el almacén principal
            else
            {
                almacenSecundario.Add(bici);
                Console.WriteLine("Un usuario en la zona de Amara devuelve la bici {0} al almacén secundario, el principal está cerrado", bici);
            }
        }
    }
```

Alquila bicis del almacén principal mientras no esté cerrado

Si el número de la bici es múltiplo de 5 devuelve la bici

Al almacén principal si no está cerrado

Al almacén secundario si el principal está cerrado

ALMACÉN PRINCIPAL y ALMACÉN SECUNDARIO de capacidad 100 bicicletas cada uno, utilizando Blocking Collection

```
// Crea la coleccion de almacenes con una capacidad máxima de 100 bicis cada uno
BlockingCollection<int> almacenPrincipal = new BlockingCollection<int>(100);
BlockingCollection<int> almacenSecundario = new BlockingCollection<int>(100);
```

Ejecución del programa:

Consola de depuración de Microsoft Visual Studio

```
El proveedor ha entregado la bici número 1 al almacén principal
Un usuario en la zona de Gros alquila la bici 1 del  almacén principal
El proveedor ha entregado la bici número 2 al almacén principal
El proveedor ha entregado la bici número 3 al almacén principal
El proveedor ha entregado la bici número 4 al almacén principal
El proveedor ha entregado la bici número 5 al almacén principal
El proveedor ha entregado la bici número 6 al almacén principal
El proveedor ha entregado la bici número 7 al almacén principal
El proveedor ha entregado la bici número 8 al almacén principal
Un usuario en la zona de Gros alquila la bici 3 del  almacén principal
Un usuario en la zona de Gros alquila la bici 4 del  almacén principal
Un usuario en la zona de Gros alquila la bici 5 del  almacén principal
Un usuario en la zona de Gros alquila la bici 6 del  almacén principal
Un usuario en la zona de Gros alquila la bici 7 del  almacén principal
Un usuario en la zona de Gros alquila la bici 8 del  almacén principal
Un usuario en la zona de Gros alquila la bici 9 del  almacén principal
Un usuario en la zona de Amara alquila la bici 2 del  almacén principal
El proveedor ha entregado la bici número 9 al almacén principal
El proveedor ha entregado la bici número 10 al almacén principal
Un usuario en la zona de Amara alquila la bici 10 del  almacén principal
El proveedor ha entregado la bici número 11 al almacén principal
El proveedor ha entregado la bici número 12 al almacén principal
El proveedor ha entregado la bici número 13 al almacén principal
El proveedor ha entregado la bici número 14 al almacén principal
El proveedor ha entregado la bici número 15 al almacén principal
Un usuario en la zona de Amara devuelve la bici 10 al almacén principal
Un usuario en la zona de Amara alquila la bici 12 del  almacén principal
Un usuario en la zona de Amara alquila la bici 13 del  almacén principal
Un usuario en la zona de Amara alquila la bici 14 del  almacén principal
Un usuario en la zona de Amara alquila la bici 15 del  almacén principal
Un usuario en la zona de Gros alquila la bici 11 del  almacén principal
Un usuario en la zona de Gros alquila la bici 10 del  almacén principal
Un usuario en la zona de Gros alquila la bici 16 del  almacén principal
Un usuario en la zona de Gros alquila la bici 15 del  almacén principal
Un usuario en la zona de Amara devuelve la bici 15 al almacén principal
El proveedor ha entregado la bici número 16 al almacén principal
El proveedor ha entregado la bici número 17 al almacén principal
```

Proveedor entrega las bicis al almacén principal

Usuario de Gros alquila bici del almacén principal

Usuario de Amara alquila bici del almacén principal

Usuario de Amara devuelve al almacén principal una bici cuyo número es múltiplo de 5

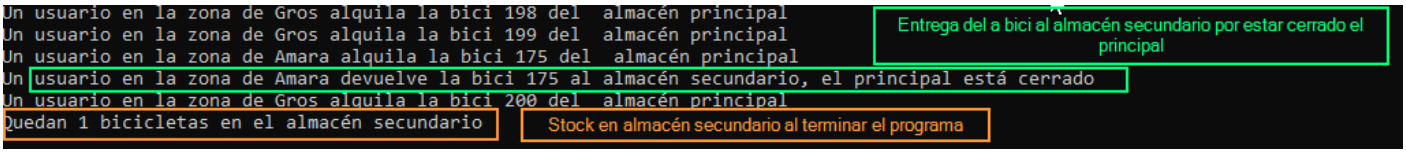
Proveedor entrega 200 bicis:

```
El proveedor ha entregado la bici número 199 al almacén principal
El proveedor ha entregado la bici número 200 al almacén principal
Un usuario en la zona de Gros alquila la bici 162 del  almacén principal
Un usuario en la zona de Gros alquila la bici 163 del  almacén principal
Un usuario en la zona de Gros alquila la bici 164 del  almacén principal
```

Cierre del almacén principal:

```
Un usuario en la zona de Gros alquila la bici 172 del  almacén principal
Un usuario en la zona de Gros alquila la bici 173 del  almacén principal
Se cierra el Almacén Principal
Un usuario en la zona de Amara alquila la bici 167 del  almacén principal
Un usuario en la zona de Gros alquila la bici 174 del  almacén principal
```

Entrega de la bici al almacén secundario por estar cerrado el principal, y stock final en el almacén secundario:



Uso de las clases Blocking Collection y Task:

```
// Crea la coleccion de almacenes con una capacidad máxima de 100 bicis cada uno
BlockingCollection<int> almacenPrincipal = new BlockingCollection<int>(100);
BlockingCollection<int> almacenSecundario = new BlockingCollection<int>(100);

// Tarea Productor: agrega 1 bici a almacenPrincipal hasta que bici = 200
Task Productor = Task.Run(() =>
{
```

Problemas encontrados:

En el ejercicio original se producían interbloqueos al concurrir los 3 en la entrega de bicis al almacén principal si este llegaba a su capacidad de 100. Quedaban a la espera de lugar, pero nadie consumía bici, por lo que el programa quedaba stand by.

Ejercicio 2

5 filósofos, cada uno con su hilo

```
int cant_filo = 5;  // 5 filosofos
Filosofo[] filosofos = new Filosofo[cant_filo];
Thread[] t = new Thread[cant_filo];

asignarPalillos(filosofos);

// crea los hilos de cada filosofo para comer
for (int i = 0; i < filosofos.Length; i++)
{
    Console.WriteLine("Creando hilo del filosofo {0}", i);
    t[i] = new Thread(filosofos[i].gestionComida);
}

// lanza los hilos
for (int i = 0; i < filosofos.Length; i++)
{
    t[i].Start();
}
```

5 palillos, y asignación de dos a cada filósofo:

```
// Asigna los palillos a cada filosofo el array
1 referencia
public static void asignarPalillos(Filosofo[] filosofos)
{

    for (int i = 0; i < 5; i++)
    {
        if (i == 0)
        {
            // el filosofo 0 es especial pues usa los palillos 4 y 0
            filosofos[i] = new Filosofo(i, filosofos.Length - 1, i, 0);
        }
        else
        {
            // los demás filosofos usan los palillos con su id - 1 y su mismo id
            filosofos[i] = new Filosofo(i, i - 1, i, 0);
        }
        Console.WriteLine("El filosofo {0} entra a cenar y utilizará los palillos: izquierdo {1} y derecho {2}",
    }
}
```

4 estados:

```
public string[] estado = { "INTENTANDO COMER", "HAMBRIENTO", "COMIENDO", "PENSANDO" }; // 4 estados;
```

El programa corre mientras los filósofos no hayan comido 5 veces:

```
Thread.Sleep(1000);
while (comido != 5)
{
    estado_aux = estado[0]; // INTENTANDO COMER
    Console.WriteLine("El filósofo {0} está {1} e intentará agarrar el
    try
    {
        // el filosofo intentará agarrar el primer palillo
        if (Monitor.TryEnter(palillos[palillo_izda], timeout))
```

Uso de la clase Thread:

```
Console.WriteLine("Creando hilo del filosofo {0}", i);
t[i] = new Thread(filosofos[i].gestionComida);
```

Uso de la clase Monitor:

```
// el filosofo intentará agarrar el primer palillo
if (Monitor.TryEnter(palillos[palillo_izda], timeout))
{
```

Ejecución el programa:

Asignación de palillos a cada filósofo, creación de un hilo para cada uno.

2 estados del filósofo 3: INTENTANDO COMER (sin palillos), HAMBRIENTO (con un palillo)

```
El filosofo 0 entra a cenar y utilizará los palillos: izquierdo 4 y derecho 0
El filosofo 1 entra a cenar y utilizará los palillos: izquierdo 0 y derecho 1
El filosofo 2 entra a cenar y utilizará los palillos: izquierdo 1 y derecho 2
El filosofo 3 entra a cenar y utilizará los palillos: izquierdo 2 y derecho 3
El filosofo 4 entra a cenar y utilizará los palillos: izquierdo 3 y derecho 4
Creando hilo del filosofo 0
Creando hilo del filosofo 1
Creando hilo del filosofo 2
Creando hilo del filosofo 3
Creando hilo del filosofo 4
El filósofo 0 está INTENTANDO COMER e intentará agarrar el palillo 4
El filósofo 3 está INTENTANDO COMER e intentará agarrar el palillo 2
El filósofo 3 agarró el palillo 2 y está HAMBRIENTO
El filósofo 2 está INTENTANDO COMER e intentará agarrar el palillo 1
```

Filósofos están COMIENDO

```
El filósofo 3 agarró el palillo 3 y está COMIENDO
El filósofo 1 está HAMBRIENTO e intentará agarrar el palillo 1
El filósofo 1 agarró el palillo 1 y está COMIENDO
El filósofo 1 ha soltado el palillo 1
```

Filósofo dejas los palillos:

```
El filósofo 2 ha soltado el palillo 2
El filósofo 2 ha soltado el palillo 1
```

Luego de comer 5 veces, filósofos pasan a estado PENSANDO:

```
El filósofo 2 ha soltado el palillo 1
El filósofo 0 ha soltado el palillo 0
El filosofo 0 está PENSANDO. Ya comió (5 veces)
El filósofo 0 ha soltado el palillo 4
```

Problemas encontrados:

A veces parece que hay un filósofo que agarra un palillo antes de que lo suelte otro, pero en realidad el mensaje por consola es lo que lleva a confusión. Si sólo monitorizamos los cambios de estado, es más fácil ver que no hay concurrencia.

3. AUTOEVALUACIÓN

Ambos programas se ejecutan correctamente y cumplen con los requisitos solicitados.

4. BIBLIOGRAFÍA

Videos de la unidad