2022

Problema Filósofos

UNIVERSIDAD POLITECNICA DE TECAMAC
FERNANDO MANUEL GALVAN

PROGRAMACION CONCURRENTE | 2722IS

Contenido

Índice de Ilustraciones

Problemática

Cinco filósofos pasan su vida pensando y comiendo. Los filósofos comparten una mesa circular rodeada por cinco sillas, una para cada uno de ellos. En el centro de la mesa se encuentra una fuente de arroz, y también sobre la mesa hay cinco palillos chinos. Cuando un filósofo piensa, no interactúa con sus colegas. Ocasionalmente, un filósofo tiene hambre y trata de coger los dos palillos que están más cerca de él (los palillos colocados entre él y sus vecinos de la derecha y de la izquierda). Un filósofo sólo puede coger un palillo a la vez y, obviamente, no puede ser el que está en la mano de un vecino. Cuando un filósofo hambriento tiene sus dos palillos al mismo tiempo come sin soltarlos. Cuando termina de comer, coloca ambos palillos sobre la mesa y comienza a pensar otra vez.

En que consiste el programa

El programa debe poder hacer que los filósofos puedan comer con palillos compartidos, sin que se bloqueen entre ellos, lo cual se usa semáforos para evitar que los valores de los palillos cambien y se pausen otros procesos a la hora de ejecutarse, a la vez existe la posibilidad de un interbloqueo lo cual es que a la hora de ejecutar el semáforo ya no pueda reanudar la operación.

Como se resolvió

Se necesito de hilos y semáforos para poder realizar el programa, primero declaramos los palillos con booleanos para que cuando estén en uso se pongan en “false” y cuando estén desocupados en “true”. Cuando los palillos que necesite el filósofo para comer estén desocupados “true” podrán comer y los palillos entraran en ocupados “false, una vez terminado el tiempo de comer los palillos se desocuparan.

Los semáforos están declarados de forma que pueden hacer dos procesos y se esperan los otros tres. El semáforo se llama al inicio de cada función con WaitOne y al final con Release.

Diseño

Ilustración 1 Diseño

Ilustración 2 Diseño "Comer"

El diseño consta de cinco pensantes alrededor de la mesa (rosquilla) en donde son representados por una imagen de homero, estos representan los filósofos de la problemática de los filósofos “Ilustración 1”.

Cuando los pensantes estén comiendo se representará con una imagen diferente y se ocultara la imagen original “Ilustración 2”.

La aplicación cuenta con un fondo amarillo para hacer contraste con los personajes y la rosquilla, para poder iniciar el programa la rosquilla funcionara como un botón.

```
public bool palillo1 = true, palillo2 = true, palillo3 = true, palillo4 = true, palillo5 = true;
public Semaphore semaforo = new Semaphore(2,3);//declaramos el semaforo
1 referencia
```

Ilustración 3 Variable

En la Ilustración 3 declaramos el semáforo, en donde le decimos que podrá ejecutar dos funciones y tres van estar en espera.

También declararemos variables booleanas que serán los palillos que usarán los filósofos y los cuales comerán. En donde si se encuentra en “true” estarán disponibles para ser usados, en caso contrario si se encuentran en “false” significa que están siendo usadas y no podrán acceder a ellas hasta que se desocupen.

```
for (int i = 0; i < 5; i++)
{
    Thread filo1 = new Thread(filosofo1);
    Thread filo2 = new Thread(filosofo2);
    Thread filo3 = new Thread(filosofo3);
    Thread filo4 = new Thread(filosofo4);
    Thread filo5 = new Thread(filosofo5);
    filo1.Start();
    filo2.Start();
    filo3.Start();
    filo4.Start();
    filo5.Start();
}
```

Ilustración 4 Hilos

En la Ilustración 4 creamos los hilos (Thread) en donde se llaman los métodos de los filósofos, y con el for hacemos que se repita 5 veces.

```
1 referencia
public void filosofo1()
{
    semaforo.WaitOne();// llamamos a l semaforo
    if (palillo1 == true && palillo2 == true)//condicion para los palillos
    {
        palillo1 = false;//ponemos a los palillos en ocupado "false"
        palillo2 = false;
        Invoke((Delegate)new Action(() => {
            filo_1.Visible = false;//cambiamos las imagenes
            hambre_1.Visible = true;
        }));
        Thread.Sleep(3000);//tiempo de espera 3 segundos
    }
    palillo1 = true;//despues de la espera
    palillo2 = true;//cambiamos los palillos a desocupado "true"
    Invoke((Delegate)new Action(() => {
        filo_1.Visible = true;
        hambre_1.Visible = false;
    }));
    semaforo.Release();// finalizacion del semaforo
}
```

Ilustración 5 filosofo 1

En la ilustración 5 se muestra la función de los filosofo1, en donde primero se inicializa el semáforo con la función WaitOne.

Luego se crea la condición en donde si el palillo 1 y el palillo 2 están desocupados “true”, realiza lo siguiente; pone en ocupados los palillos 1 y 2 “false”.

Luego usamos la función “Invoke” para modificar las imágenes del Form ya que un subproceso no puede acceder a ellas y modificarlas lo cual es necesario esta función.

Siguiente de esto le damos un tiempo de espera con la función de Thread.Sleep() en donde le asignamos “3000” milisegundos que son un equivalente de tres segundos.

Después de que comieran los filósofos pasando los tres segundos, cambiamos los palillos a “true” donde se desocupan, poniéndolos de nuevo en desocupados.

Cambiamos las imágenes con la función Invoke, para luego poder finalizar el semáforo con la función Release y evitar un bloqueo.

```
1 referencia
public void filosofo2()
{
    semaforo.WaitOne();
    if (palillo2 == true && palillo3 == true)
    {
        palillo2 = false;
        palillo3 = false;
        Invoke((Delegate)new Action(() => {
            filo_2.Visible = false;
            hambre_2.Visible = true;
        }));
        Thread.Sleep(3000);
    }
    palillo2 = true;
    palillo3 = true;
    Invoke((Delegate)new Action(() => {
        filo_2.Visible = true;
        hambre_2.Visible = false;
    }));
    semaforo.Release();
}
```

Ilustración 6 filosofo 2

```
1 referencia
public void filosofo3()
{
    semaforo.WaitOne();
    if (palillo3 == true && palillo4 == true)
    {
        palillo3 = false;
        palillo4 = false;
        Invoke((Delegate)new Action(() => {
            filo_3.Visible = false;
            hambre_3.Visible = true;
        }));
        Thread.Sleep(3000);
    }
    palillo3 = true;
    palillo4 = true;
    Invoke((Delegate)new Action(() => {
        filo_3.Visible = true;
        hambre_3.Visible = false;
    }));
    semaforo.Release();
}
```

Ilustración 7 filosofo 3

```
1 referencia
public void filosofo4()
{
    semaforo.WaitOne();
    if (palillo4 == true && palillo5 == true)
    {
        palillo4 = false;
        palillo5 = false;
        Invoke((Delegate)new Action(() => {
            filo_4.Visible = false;
            hambre_4.Visible = true;
        }));
        Thread.Sleep(3000);
    }
    palillo4 = true;
    palillo5 = true;
    Invoke((Delegate)new Action(() => {
        filo_4.Visible = true;
        hambre_4.Visible = false;
    }));
    semaforo.Release();
}
```

Ilustración 8 filosofo 4

```
1 referencia
public void filosofo5()
{
    semaforo.WaitOne();
    if (palillo5 == true && palillo1 == true)
    {
        palillo5 = false;
        palillo1 = false;
        Invoke((Delegate)new Action(() => {
            filo_5.Visible = false;
            hambre_5.Visible = true;
        }));
        Thread.Sleep(3000);
    }
    palillo5 = true;
    palillo1 = true;
    Invoke((Delegate)new Action(() => {
        filo_5.Visible = true;
        hambre_5.Visible = false;
    }));
    semaforo.Release();
}
```

Ilustración 9 filosofo 5

Todas las funciones de filósofos 1, 2, 3, 4 y 5 tienen las mismas características, lo que cambian son los palillos que usaran; el filósofo 1 usara los palillos 1 y 2, el filósofo 2 usara los palillos 2 y 3, el filósofo 3 usara los palillos 3 y 4, el filósofo 4 usara los palillos 4 y 5, el filósofo 5 usara los palillos 5 y 1.

Para que pueda comer un filosofo los palillos que necesita deben estar desocupados, si no, no podrá comer hasta que nadie los esté usando.

Ejecución

Pantalla normal sin ejecutar.

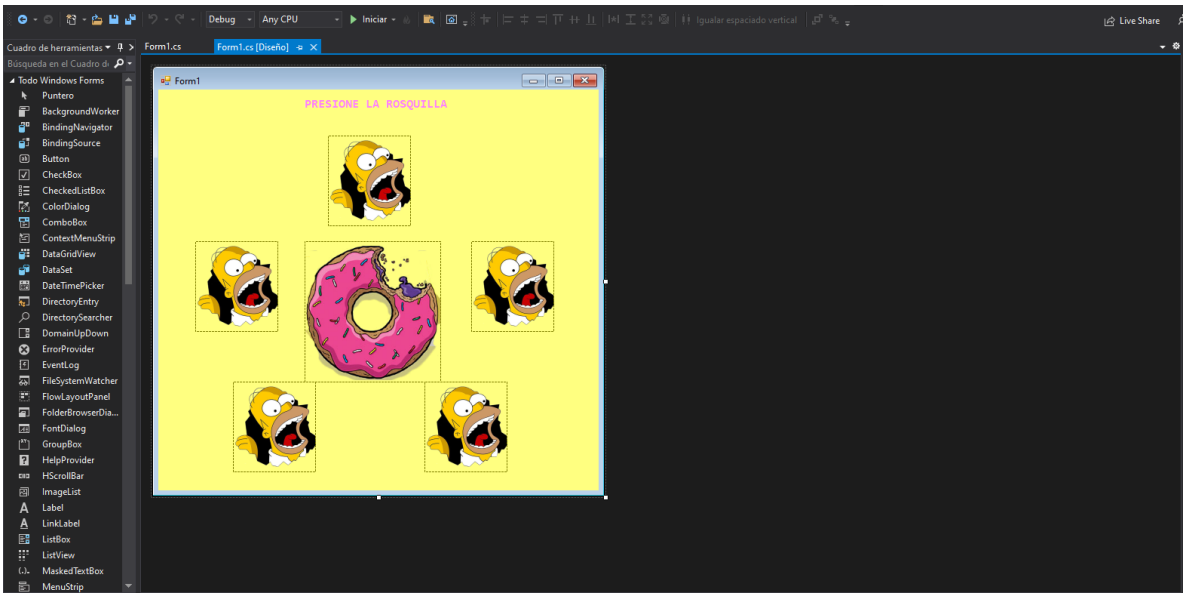

Ilustración 10 programa "1"

Programa ya ejecutado, sin realizar su proceso de comer.

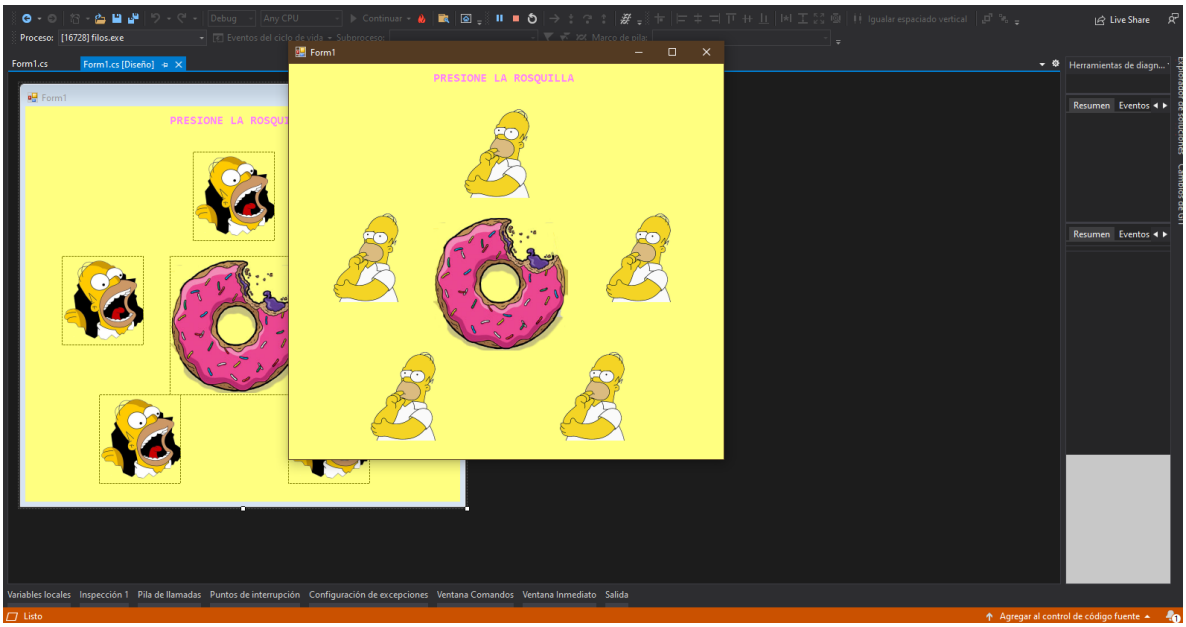

Ilustración 11 programa "2"

Programa realizando la función de pensar y comer de los filósofos.

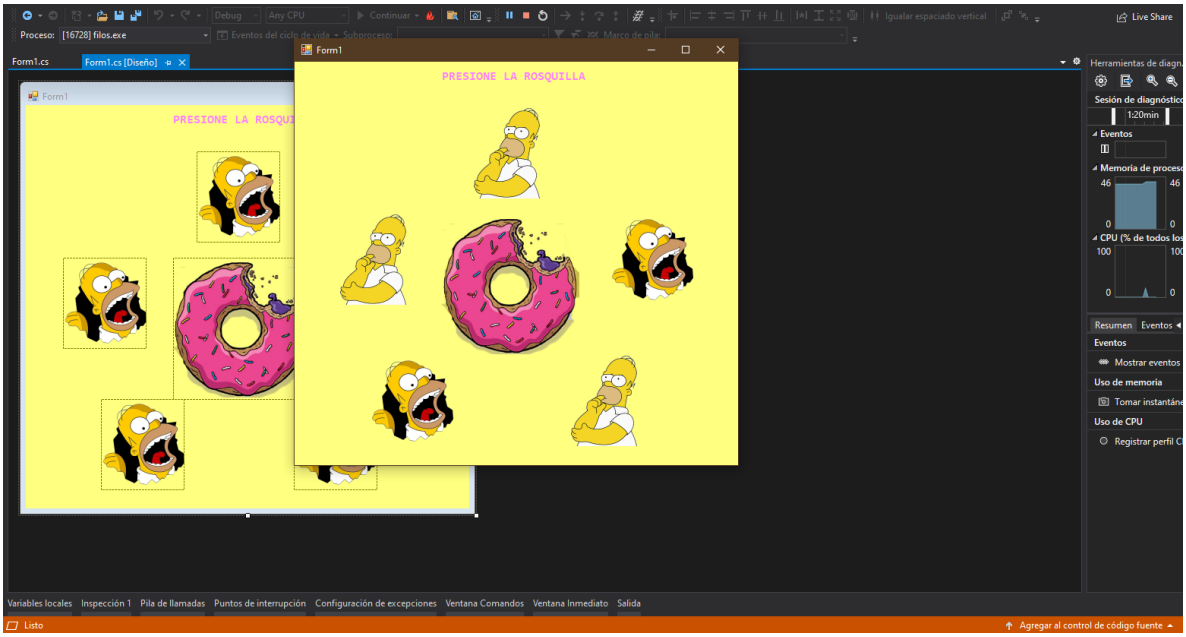

Ilustración 12 programa "3"

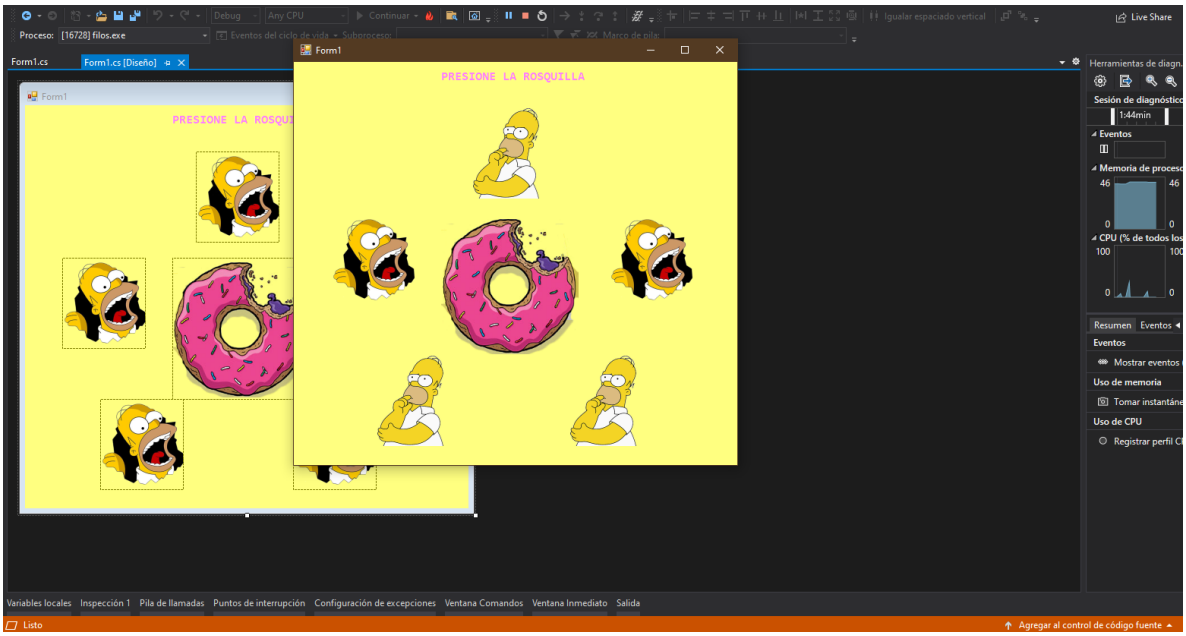

Ilustración 13 programa "4"

Como subir un repositorio a GitHub con consola

Se explicará como subir un proyecto en GitHub a través de consola.

- Primero debemos crear el repositorio en GitHub
- Luego copiamos el link que nos dan para clonarlo en nuestro dispositivo
- Para ello abrimos la consola e insertamos el comando git clone “link copiado”
- Con esto ya creo conexión con github ahora ponemos todos los documentos que deseamos subir
- Luego de poner todos los documentos abrimos la consola y ponemos el siguiente comando “git status” para ver los nuevos archivos
- Después nos mostrar los archivos nuevos o editados y ejecutamos el siguiente comando “git add .” para agregar los cambios.
- Después usamos “git commit -m '' para agregar un comentario
- Y por último usamos “git push” para guardar los cambios.

Conclusiones

Para poder realizar esta problemática fueron necesarias dos cosas importantes hilos y semáforos, los hilos se utilizan para poder realizar varias tareas a la vez, como en el ejercicio tenemos que compartir variables podemos tener corrupción de datos en donde dos métodos pueden usar la misma variable y no se esperan para que uno lo use y después otro lo cual causa la corrupción.

Para ello sirven los semáforos para indicar que si un proceso va usar cierta variable, otro proceso que la necesite se espere a que termine de usarla y no la modifique antes de que termine el proceso. Pero también al usar un semáforo se puede bloquear y nunca salir de la espera que puso al otro proceso, por eso es necesario indicar cuando un semáforo debe de terminar para evitar el bloqueo.

Con esto nos podemos dar cuenta de las ventajas y desventajas que conllevan los hilos y semáforos. Dándonos una idea de como tener que utilizarlos y sus consecuencias si no son bien estructuradas.