

12-10-2022

Rubrica acerca del proyecto de los filósofos

programación concurrente



UNIVERSIDAD POLITÉCNICA
DE TECÁMAC

ESTUDIANTE EDUARDO ANTONIO MENDEZ
SANDOVAL
2722IS

INDICE

TABLA DE ILUSTRACIONES	2
DESCRIPCIÓN DEL PROBLEMA	3
EXPLICACIÓN DEL CODIGO CON IMAGENES	4
Interfaz del proyecto.....	4
librerías	4
Cubiertos y semáforo	4
Hilos.....	5
Hilo del primer personaje	5
Hilo del segundo personaje.....	6
EJECUCIÓN DEL PROGRAMA.....	6
Primera etapa	6
Segunda etapa	7
Segunda etapa: segunda prueba.....	7
Como subir el proyecto a un repositorio	8
MAPA SOBRE LA DIFERENCIA DE PROGRAMACIÓN SECUENCIAL Y CONCURRENTE.....	9
RUBRICA DEL MAPA	10
Conclusión.....	11

TABLA DE ILUSTRACIONES

Figura 1. diseño del proyecto.....	4
Figura 2. librerías	4
Figura 3. declaración de cubiertos y semáforo	4
Figura 4. declarar hilos e iniciarlos	5
Figura 5. hilo del personaje 1	5
Figura 6. hilo del segundo y demás personajes	6
Figura 7. primera etapa de la ejecución	6
Figura 8. Etapa 2 prueba 1 de la ejecución	7
Figura 9. etapa 2 prueba 2 de la ejecución.....	7
Figura 10. subir los archivos al repositorio por medio de git imagen 2.....	8
Figura 11. subir los archivos al repositorio por medio de git	8
Figura 12. mapa acerca de la diferencia de la programación secuencial y concurrente	9
Figura 13. rúbrica del mapa.....	10

DESCRIPCIÓN DEL PROBLEMA

En este documento se llevo acabo el proyecto acerca de los filósofos, esta practica consta de 5 personajes, los cuales se encuentran con un plato cada uno, el problema consta en que los personajes necesitan 2 cubiertos para poder comer el plato que tienen delante y en la mesa se encuentran 5 cubiertos, por lo que solo 2 personajes podrán comer mientras los otros 3 personajes deberán de esperar su turno para que los cubiertos dejen de ser ocupados y estos puedan tomar sus dos cubiertos para poder comer.

Para comenzar debemos tener en cuenta que es la programación concurrente y esta no es más que la forma en la cual podemos resolver ciertas problemáticas de forma concurrente, es decir, ejecutando múltiples tareas a la misma vez y no de forma secuencial. En un programa concurrente las tareas pueden continuar sin la necesidad que otras comiencen o finalicen.

Los semáforos son un mecanismo de sincronización de procesos permiten al programador que el proceso que se está llevando a cabo no sea interrumpido, pero este debe de tener un tiempo para ocupar una variable compartida, mientras que el segundo proceso se tiene que esperar a que la variable sea desocupada por el proceso que la esté ocupando, esto nos ayuda bastante para evitar la corrupción de datos, el cual puede ser interpretado como datos que no dan el resultado correcto debido a que la variable fue adquirida por muchos procesos y esta se modificó a cada rato, dando un resultado incorrecto .

EXPLICACIÓN DEL CODIGO CON IMAGENES



Figura 1. diseño del proyecto

Interfaz del proyecto

Este es el diseño del programa, consta con 5 personajes, los cuales deberán de acceder a dos palillos, solo 2 personajes podrán comer y los otros tendrán que esperar, el programa cuenta con un botón el cual al pulsar

comenzarán los personajes a comer.

librerías

Las librerías que se ocuparon son las que nos da por defecto el programa, solamente agregamos la librería del “Threading” para la creación de los hilos.

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Threading;
9 using System.Threading.Tasks;
10 using System.Windows.Forms;
11
```

Figura 2. librerías

```
3 referencias
public partial class Form1 : Form
{
    //declarar los 5 cubiertos
    public bool cubierto1 = true, cubierto2 = true, cubierto3 = true, cubierto4 = true, cubierto5 = true;
    //declaramos la funcion del semaforo
    public Semaphore sem = new Semaphore(2, 3);
    1 referencia
}
```

Figura 3. declaración de cubiertos y semáforo

Cubiertos y semáforo

En esta parte del código vamos a declarar las variables de los palillos y el semáforo, diciendo que dos palillos van a funcionar y los otros 3 tendrán que esperar.

```

1 referencia
private void button1_Click(object sender, EventArgs e)
{
    //Ingresamos los hilos de los personajes y los le damos inicio a los hilos
    for (int i = 0; i < 5; i++)
    {
        Thread adanco = new Thread(new ThreadStart(adan));
        Thread budaco = new Thread(buda);
        Thread vegetaco = new Thread(vegeta);
        Thread hadesco = new Thread(hades);
        Thread gokuco = new Thread(goku);
        adanco.Start();
        budaco.Start();
        vegetaco.Start();
        hadesco.Start();
        gokuco.Start();
    }
}

```

Figura 4. declarar hilos e iniciarlos

Hilos

En este punto declaramos los hilos de los 5 personajes, colocamos el Thread y el nombre del hilo, después colocamos que es igual a un nuevo hilo e iniciamos el hilo que corresponde a la función

del personaje, cada hilo que se creo tiene el nombre del personaje con una palabra “co”, para que no sea el mismo nombre de la función del personaje, en este caso se aplicó un for para que el programa se repita 5 veces, cada uno deberá de comer 5 veces para que el programa finalice.

Hilo del primer personaje

El hilo de este personaje consta del nombre de este, para posteriormente iniciar el semaforo con el WaitOne(), despues declaramos un if con los cubiertos que el personaje tiene a su disposicion, el if nos dice que si el cubierto5 y el cubierto 1

```

public void adan()
{
    //declaramos el semaforo
    sem.WaitOne();
    //declaramos los cubiertos que el personaje va a ocupar
    if (cubierto5 == true && cubierto1 == true) {
        //declaramos las variables de cubiertos como false
        cubierto1 = false;
        cubierto5 = false;
        Invoke((Delegate)new Action() => {
            //si los palillos no estan ocupados, el personaje los tomara
            Adan.Visible = false;
            AdanComiendo.Visible = true;
        });
        //dormimos el hilo por 5 segundo
        Thread.Sleep(5000);
    }
    Invoke((Delegate)new Action() => {
        //cuando acaben los 5 segundos el personaje dejara las variables para que otro personaje las e
        Adan.Visible = true;
        AdanComiendo.Visible = false;
    });
    cubierto1 = true;
    cubierto2 = true;
    sem.Release();
}

```

Figura 5. hilo del personaje 1

no tienen un valor 0, entonces declaramos los cubiertos como false, para despues utilizar el Invoke el cual es un subproceso que sirve para poder hacer la modificacion del forms, diciendole que la imagen del personaje norma no se va a mostrar en el form, pero la imagen del personaje comiendo si me mostrara significando que este se encuentra ocupando los cubiertos que estas a su disposicion.

```

public void buda()
{
    sem.WaitOne();
    if (cubierto1 == true && cubierto2 == true)
    {
        cubierto1 = false;
        cubierto2 = false;
        Invoke((Delegate)new Action() => {
            Buda.Visible = false;
            BudaComiendo.Visible = true;
        });
        Thread.Sleep(5000);
    }
    Invoke((Delegate)new Action() => {
        Buda.Visible = true;
        BudaComiendo.Visible = false;
    });
    cubierto2 = true;
    cubierto3 = true;
    sem.Release();
}

```

Figura 6. hilo del segundo y demás personajes

Hilo del segundo personaje

Al igual que el anterior personaje, este va a tener la misma función, lo único que cambia son los cubiertos que este utiliza, este al tener que ocupar el cubierto número 1, debe de esperar el momento

en el que el primer personaje lo deje de ocupar, este personaje cuenta con los cubiertos 1 y 2 debido a que estos son los más cercanos a él, al final de la función reiniciamos el semáforo, debido a que si no lo hacemos este nos mandara un error, de aquí en adelante los hilos de los demás personajes son iguales, lo único que cambia es que el 3er personaje tiene a su alcance el cubierto 2 y 3, el 4to personaje tiene a su alcance el cubierto 3 y 4, el 5to personaje tiene a su alcance los cubiertos 4 y 5.

EJECUCIÓN DEL PROGRAMA

Primera etapa

Como se puede apreciar el programa no presenta ningún error y las imágenes de los personajes que se muestran son normales, debido a que los hilos todavía no se corren, para que el programa comience a funcionar las imágenes de dos personajes deberán cambiar dando como significado que están ocupando los cubiertos.

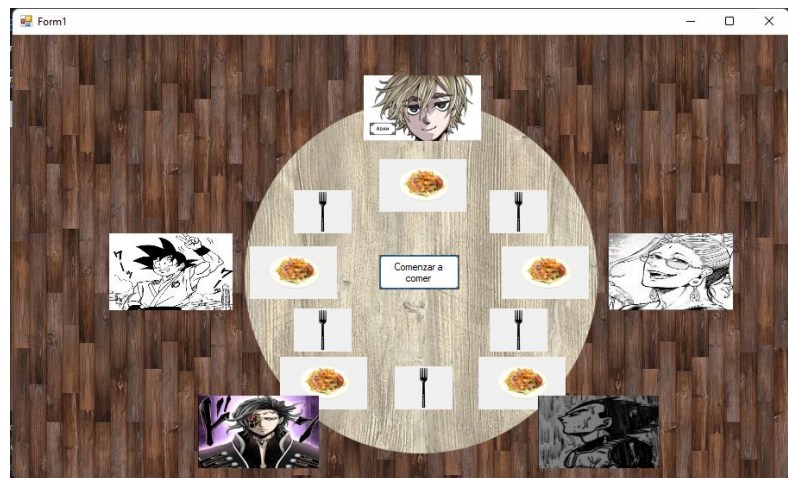


Figura 7. primera etapa de la ejecución



Figura 8. Etapa 2 prueba 1 de la ejecución

Segunda etapa

Esta segunda etapa ocurre cuando se pulsa el boton, el cual corre los 5 hilos de los personajes y dependiendo de los dos primeros que tomen los cubiertos los demas deberan de esperar a que terminen de ocuparlos para poder comer, sin embargo si un personaje se llega a tardar en tomar los cubiertos el

personaje que los tenia anteriormente los puede volver a tomar y el otro personaje debera de esperarse de nuevo hasta que este desocupe los cubiertos.

Segunda etapa: segunda prueba

En esta segunda prueba se puede observar como el personaje 1, Adan, y el personaje 4, Hades, dejaron de ocupar los cubiertos 1, 4 y 5 para que ahora el personaje 2, Buda, y el personaje 5, Goku, puedan comer ocupando los cubiertos que dejaron de ocupar.

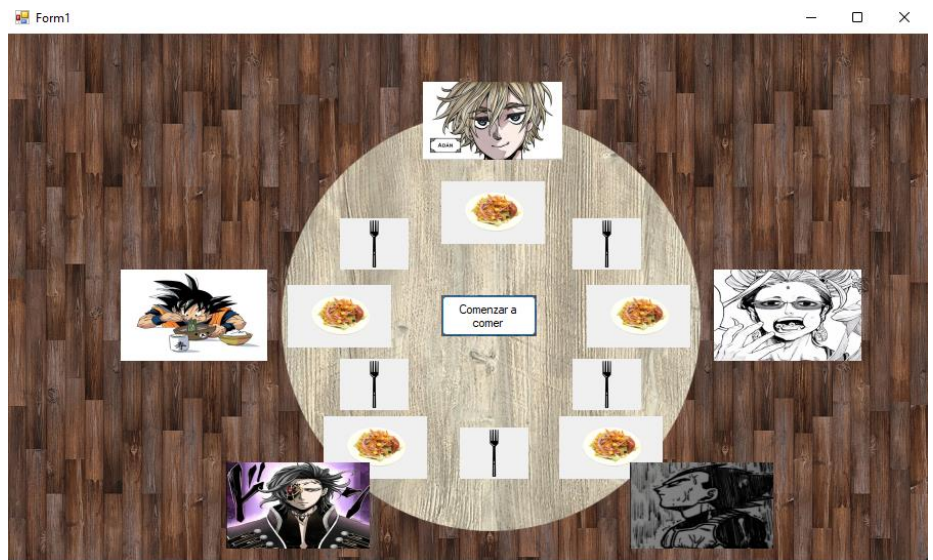
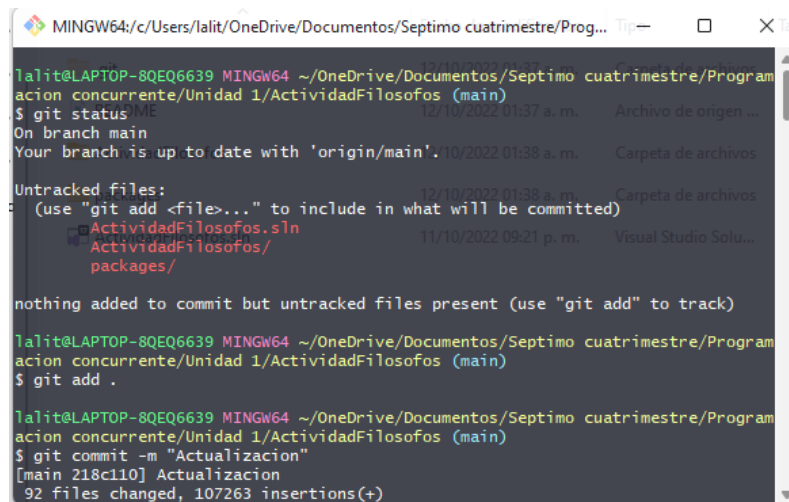


Figura 9. etapa 2 prueba 2 de la ejecución

Como subir el proyecto a un repositorio



```
MINGW64/c/Users/lalit/OneDrive/Documentos/Septimo cuatrimestre/Program...
lalit@LAPTOP-8QE6639 MINGW64 ~/OneDrive/Documentos/Septimo cuatrimestre/Program
acion concurrente/Unidad 1/ActividadFilosofos (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  ActividadFilosofos.sln
  ActividadFilosofos/
  packages/

nothing added to commit but untracked files present (use "git add" to track)
lalit@LAPTOP-8QE6639 MINGW64 ~/OneDrive/Documentos/Septimo cuatrimestre/Program
acion concurrente/Unidad 1/ActividadFilosofos (main)
$ git add .
lalit@LAPTOP-8QE6639 MINGW64 ~/OneDrive/Documentos/Septimo cuatrimestre/Program
acion concurrente/Unidad 1/ActividadFilosofos (main)
$ git commit -m "Actualizacion"
[main 218c110] Actualizacion
92 files changed, 107263 insertions(+)
```

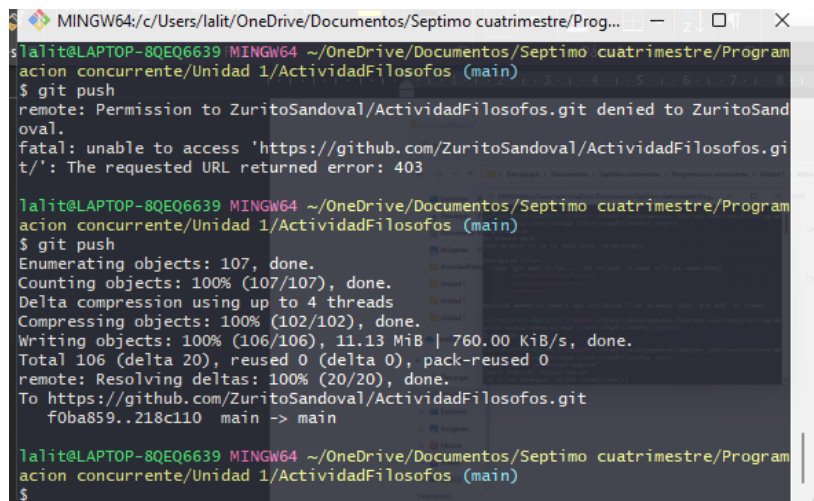
Figura 11. subir los archivos al repositorio por medio de git

Para lograr subir el proyecto a un repositorio, primero se tuvo que crear el repositorio en github, despues se tuvo que clonar el repositorio en una carpeta de la computadora con el comando “git clone”, una vez terminada la clonacion se copiara los

documentos del proyecto y se pegara en la carpeta que clonamos, ahí utilizaremos el comando “gti status” para que reconozca los archivos del proyecto que pegamos, despues utilizaremos el comando “git add .”

Una vez que termine el comando “git add .” se procedera a poner el comando “git committ -m “ACtualizacion”” esto para que los archivos que se suban tengan como nombre “Actualizacion”, una vez que termines con

ese paso, terminaremos con el comando “git push” para subir todos los archivos a nuestro repositorio, recargamos la pagina y veremos que los documento se subieron exitosamente.



```
MINGW64/c/Users/lalit/OneDrive/Documentos/Septimo cuatrimestre/Program...
lalit@LAPTOP-8QE6639 MINGW64 ~/OneDrive/Documentos/Septimo cuatrimestre/Program
acion concurrente/Unidad 1/ActividadFilosofos (main)
$ git push
remote: Permission to ZuritoSandoval/ActividadFilosofos.git denied to ZuritoSand
oval.
fatal: unable to access 'https://github.com/ZuritoSandoval/ActividadFilosofos.git/': The requested URL returned error: 403

lalit@LAPTOP-8QE6639 MINGW64 ~/OneDrive/Documentos/Septimo cuatrimestre/Program
acion concurrente/Unidad 1/ActividadFilosofos (main)
$ git push
Enumerating objects: 107, done.
Counting objects: 100% (107/107), done.
Delta compression using up to 4 threads
Compressing objects: 100% (102/102), done.
Writing objects: 100% (106/106), 11.13 MiB | 760.00 KiB/s, done.
Total 106 (delta 20), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (20/20), done.
To https://github.com/ZuritoSandoval/ActividadFilosofos.git
 f0ba859..218c110 main -> main

lalit@LAPTOP-8QE6639 MINGW64 ~/OneDrive/Documentos/Septimo cuatrimestre/Program
acion concurrente/Unidad 1/ActividadFilosofos (main)
$
```

Figura 10. subir los archivos al repositorio por medio de git imagen 2

MAPA SOBRE LA DIFERENCIA DE PROGRAMACIÓN SECUENCIAL Y CONCURRENTE

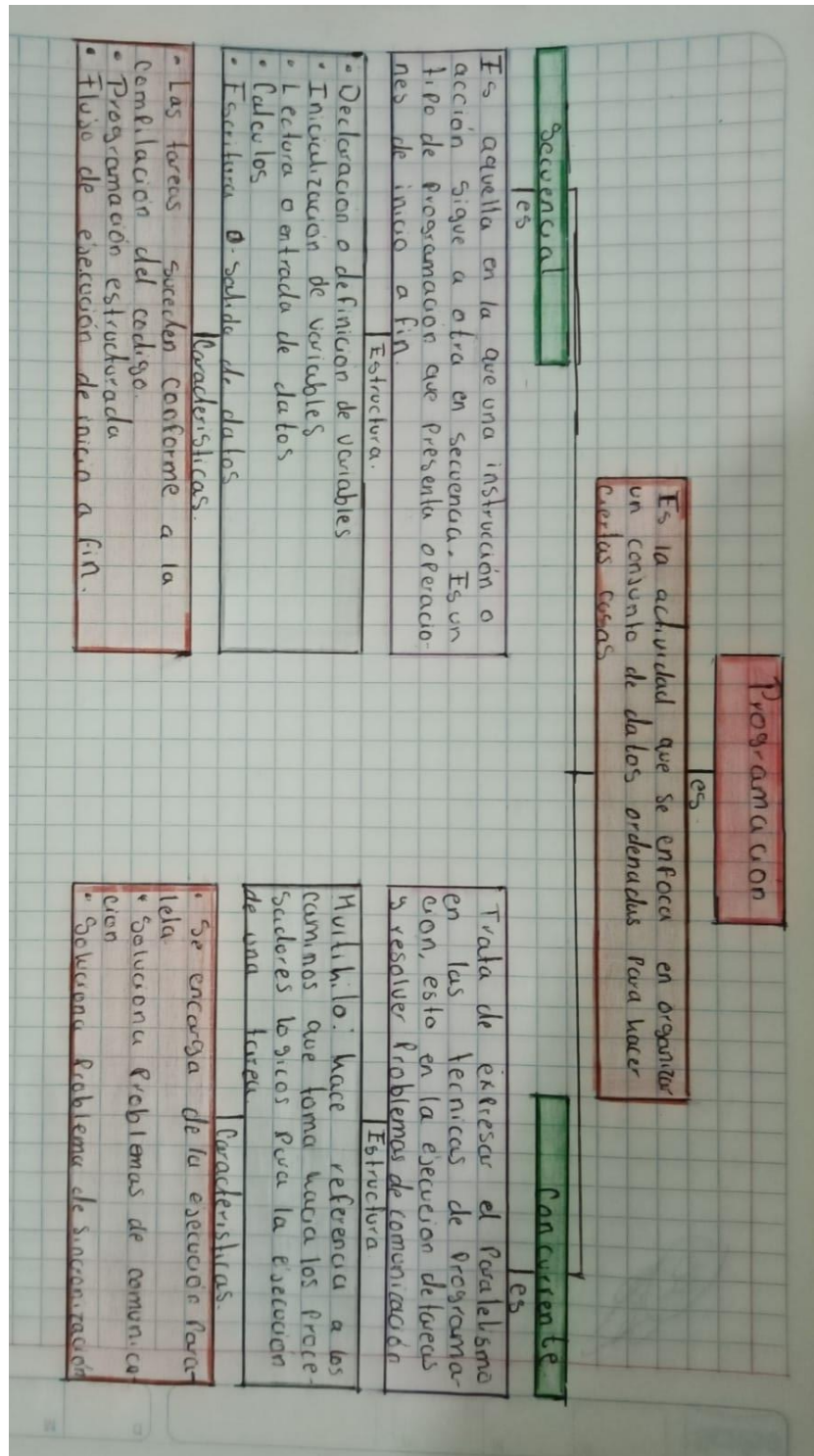

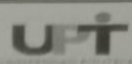


Figura 12. mapa acerca de la diferencia de la programación secuencial y concurrente

RUBRICA DEL MAPA

		LISTA DE COTEJO MAPA CONCEPTUAL PROGRAMACION CONCURRENTE Y SECUENCIAL UNIDAD 1			
DATOS GENERALES DEL PROCESO DE EVALUACIÓN					
NOMBRE DEL ALUMNO: <i>Mendez Sandoval Eduardo Antonio</i>					
MATRICULA: <i>1320114070</i>			FECHA: 9 SEPTIEMBRE 2021		
NOMBRE DEL PRODUCTO: MAPA MENTAL PROGRAMACION CONCURRENTE Y SECUENCIAL					
NOMBRE DE LA ASIGNATURA: PROGRAMACIÓN CONCURRENTE			CUATRIMESTRE O CICLO DE FORMACIÓN: SÉPTIMO CUATRIMESTRE		
NOMBRE DEL DOCENTE: BENITO RAMIREZ FUENTES					
Instrucciones: Realizar un mapa conceptual sobre las diferencias entre la programación concurrente y la secuencial implementación semáforo					
INSTRUCCIONES					
Revisar las actividades que se solicitan y marque en los apartados "SI" cuando la evidencia se cumple; en caso contrario marque "NO". En la columna "OBSERVACIONES" indicaciones que puedan ayudar al alumno a saber cuáles son las condiciones no cumplidas, si fuese necesario.					
VALOR	REACTIVO	CUMPLE	OBSERVACIONES		
40%	CLARIDAD DE LOS CONCEPTOS	✓			
20%	USO DE IMÁGENES Y COLORES	✓			
5%	USO DEL ESPACIO LINEAS Y TEXTOS.	✓			
10%	ENFASIS Y ASOCIACIONES.	✓			
5%	SIN ERRORES ORTOGRAFICOS	✓			
20%	RELACIONA CON LA MATERIA	✓			
100%		CALIFICACIÓN	<i>100</i>		

[Firma]

Figura 13. rúbrica del mapa

Conclusión

Esta practica nos sirvio como una manera practica de entender como es que funcionan los hilos y como funcionan los semaforos y la importancia de estos al momento de tener mas de dos hilos, debido a que este puede provocar una corrupcion de datos no deseada, claro esto depende de cada computadora, debido a que el tiempo que se le asigno al Thread.Sleep fue de 5 segundos, pero esto puede variar debido al procesador de cada computadora.

También gracias al desarrollo de esta práctica se puede entender más a fondo el tema de programación concurrente, ya que esta no es más que la forma en la cual podemos resolver ciertas problemáticas de forma concurrente, es decir, ejecutando múltiples tareas a la misma vez y no de forma secuencial.

En un programa concurrente las tareas pueden continuar sin la necesidad que otras comiencen o finalicen, en este caso los dos personajes podían comer al mismo tiempo sin importar si uno ya había terminado o seguía comiendo, claro que el semáforo ayudo a que el personaje tuviera un tiempo determinado para ocupar cada cubierto.

Si bien es cierto que la programación concurrente acarrea ciertos problemas, principalmente al momento de compartir información entre tareas, también es cierto que, si se implementa de forma correcta, podremos, en casos puntuales, mejorar significativamente el performance de nuestras aplicaciones, el objetivo de la práctica no se obtuvo como se quería, pero si se reforzo los temas que no se tenían tan claros al inicio del proyecto.