

Rubrica

Universidad
politécnica de
Tecámac



Unidad I

Nombre: Fernando Brayan Mejía Gómez

Matricula: 1320114054

Grupo: 2722 IS

Séptimo cuatrimestre

Ingeniería en software

Asignatura: Programación concurrente

Profesor: Benito Ramírez Fuentes

Índice

Introducción.....	4
Problemática.....	5
Diseño del programa.....	6
Desarrollo del programa.....	8
Pruebas.....	12
Subida de GitHub	14
Mapa conceptual	16
Rubrica mapa conceptual	17
Conclusiones.....	18

Tabla de figuras

Figura 1 Ejemplo de la mesa de filósofos.....	5
Figura 2 Diseño en C#	6
Figura 3 Propiedades del forms	6
Figura 4 Propiedades del avatar	7
Figura 5 Propiedad de avatar 2	7
Figura 6 Variables publicas	8
Figura 7 Threads con parámetros	9
Figura 8 Función de parámetros	9
Figura 9 Semáforo inicialización.....	10
Figura 10 En otro caso del if.....	10
Figura 11 Verificar el estado de los palillos	11
Figura 12 Finalización de semáforo	11
Figura 13 Form de la ventana	12
Figura 14 Pruebas de escritorio	12
Figura 15 Pruebas de escritorio 2	13

Introducción

La programación concurrente ha sido sin lugar a duda uno de los grandes pilares para la programación ya que mediante esta se ha logrado un gran avance para las consultas y registros simultáneos en menor tiempo, ya que funciona a través de algo llamado threads o en español conocidos como hilos, estos funcionan gracias a la cantidad de núcleos que ofrece nuestro CPU (central processing unit) el cual hace uso de cada uno de los caminos hacia los procesadores lógicos para suministrar mejor el tiempo de ejecución de un programa. Pero la programación concurrente se encontró con varios problemas los cuales son deadlock y corrupción de datos, estas dos características son propiciadas a la mala administración de recursos (variables) estas pueden tomar una cada proceso y si un proceso toma una variable que necesitaba otro proceso y se quede en espera de la misma infinitamente, se le conoce como deadlock, la corrupción de datos se ocasiona cuando un proceso tarda mas de lo esperado y modifica el dato que se hizo en otro de los hilos, esto puede ocasionar una diferencia abismal de los resultados esperados.

En este documento se aborda la problemática de los 5 filósofos sinodales que se encuentran en una mesa para comer, sin embargo solo dos de ellos podrán comer y los demás quedaran en espera hasta que suelten los cubiertos para poder comer, estos tiene que merendar casi la misma cantidad de comida, en este documento se aborda el programa hecho mediante el lenguaje de programación c# de manera visual (forms) estos comerán 5 repeticiones en las cuales solo dos de ellos merendarán más que los demás, esto resuelto a través de la programación concurrente, utilizando estructuras de datos y de secuencia, una de las funciones importantes para el funcionamiento de estas es semáforos que permitirán ejecutar 2 procesos a la vez haciendo que 3 entren en espera.

Problemática

Cinco filósofos se sientan alrededor de una mesa y pasan su vida cenando y pensando. Cada filósofo tiene un plato de fideos y un tenedor a la izquierda de su plato. Para comer los fideos son necesarios dos tenedores y cada filósofo sólo puede tomar los que están a su izquierda y derecha. Si cualquier filósofo toma un tenedor y el otro está ocupado, se quedará esperando, con el tenedor en la mano, hasta que pueda tomar el otro tenedor, para luego empezar a comer.

Si dos filósofos adyacentes intentan tomar el mismo tenedor a una vez, se produce una condición de carrera: ambos compiten por tomar el mismo tenedor, y uno de ellos se queda sin comer.

Si todos los filósofos toman el tenedor que está a su derecha al mismo tiempo, entonces todos se quedarán esperando eternamente, porque alguien debe liberar el tenedor que les falta. Nadie lo hará porque todos se encuentran en la misma situación (esperando que alguno deje sus tenedores). Entonces los filósofos se morirán de hambre. Este bloqueo mutuo se denomina interbloqueo o *deadlock*.

El problema consiste en encontrar un algoritmo que permita que los filósofos nunca se mueran de hambre.

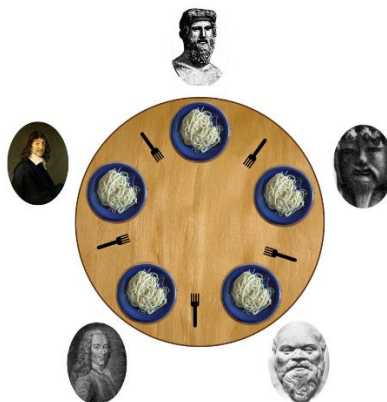


Figura 1 Ejemplo de la mesa de filósofos

Diseño del programa.

Para el diseño del programa se hizo en un forms de C# el cual contiene una mesa redonda con una imagen PNG, la cual contiene dentro de ella 5 palillos en formato PNG donde cada filósofo (en este caso es un personaje de una caricatura) podrá tomar sus palillos respectivo. En el centro de la mesa tiene un botón que permitirá inicializar la comida de los filósofos dentro de este contiene los hilos que se ejecutaran en nuestro programa. También hace falta recalcar que las imágenes de los filósofos comiendo se encuentran en el mismo lugar con los respectivos nombres “bobsponja#numerolugar” y “bobtoronja#numerodelugar” donde bobsponja es el filósofo en su estado normal no comiendo y bobtoronja es el estado donde se ve comiendo, a través de la propiedad de visibilidad se fueron intercalando la comida de estos.

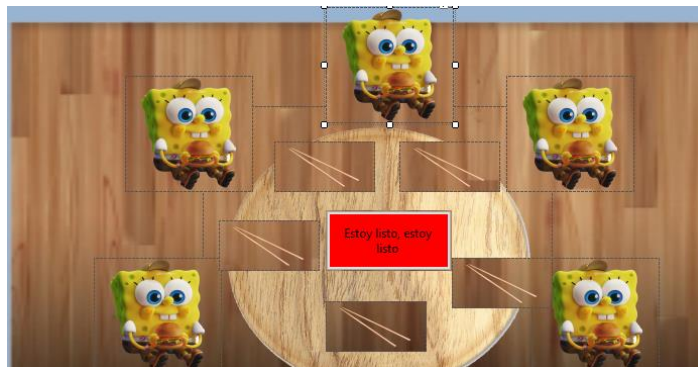


Figura 2 Diseño en C#

Aquí las propiedades del forms en general, donde se especifico el fondo que contendría el panel forms.

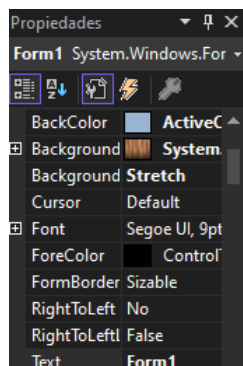


Figura 3 Propiedades del forms

Las propiedades de los avatares (filosofos) que se ocuparon dentro del programa, se especificaron la transparencia de la imagen, así como también el nombre de los avatares y palillos, esta configuración se aplicó para todos los elementos visibles dentro del forms.

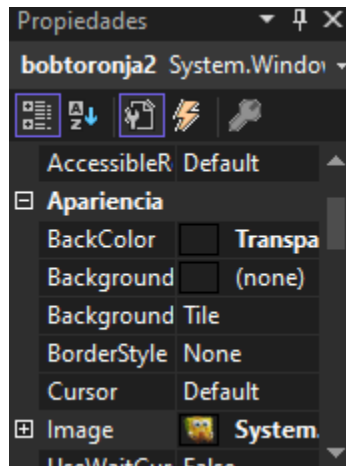


Figura 4 Propiedades del avatar

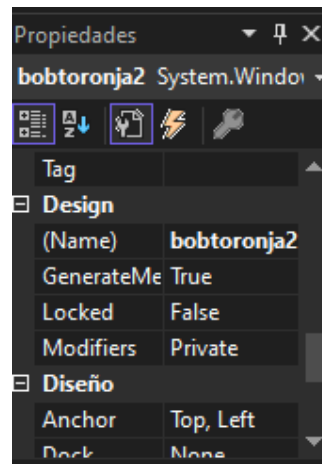


Figura 5 Propiedad de avatar 2

Desarrollo del programa.

Dentro del programa se hizo de dos maneras diferentes mediante 5 funciones diferentes donde se ejecuta cada uno de los filósofos según su respectivo lugar para comer y hacer su respectivo resultado, y la otra manera es de mandar parámetros a una función para su ejecución.

Primeramente, es necesario declarar variables publicas las cuales son de tipo int y bool, las cuales desarrollan las siguientes funciones.

Tipo	Nombre	Explicación
Bool	Palillo 1 al 5	Sirve para darle un valor de ocupación a los palillos y si es posible poder acceder a él o no.
Int	Filósofo y palillos	Estos sirven para especificar el número de filósofos, así como también el número de palillos que existen en la mesa la cual es la misma que los filósofos por tanto tenemos que: FILOSOFO==PALILLOS

```
namespace filosofos
{
    4 referencias
    public partial class Form1 : Form
    {
        public bool palillo1 = true, palillo2 = true, palillo3 = true, palillo4 = true, palillo5 = true;
        public static int filosofo = 5;
        public int palillos = filosofo;
        public Semaphore sem = new Semaphore(2,3);
        1 referencia
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Figura 6 Variables publicas

Para la función que recibirá nuestro botón escribiremos los hilos que recibirán para la ejecución del programa en la cual reciben parámetros para la función, este parámetro es de tipo entero, ya que especificara el número del filosofo para poder acceder al proceso que se llevara a cabo, esto con el fin de evitar la corrupción de datos y evitar también un posible deadlock.


```

1 referencia
private void button1_Click(object sender, EventArgs e)
{
    //Hilos con parametros
    Thread hbobspanja = new Thread(new ThreadStart(() => parametros(1)));
    hbobspanja.Start();
    Thread hbobspanja2 = new Thread(new ThreadStart(() => parametros(2)));
    hbobspanja2.Start();
    Thread hbobspanja3 = new Thread(new ThreadStart(() => parametros(3)));
    hbobspanja3.Start();
    Thread hbobspanja4 = new Thread(new ThreadStart(() => parametros(4)));
    hbobspanja4.Start();
    Thread hbobspanja5 = new Thread(new ThreadStart(() => parametros(5)));
    hbobspanja5.Start();
}

```

Figura 7 Threads con parámetros

```

public void parametros(int filosofo)
{
    //Palillos inicializacion
    int Pizq = filosofo + 1;
    int Pd = filosofo;
    //si es filosofo 5 se modifica palillo izquierdo para 1
    if (filosofo == 5)
    {
        Pizq = 1;
    }
    //inicializar semaforo
    sem.WaitOne();
    if(Pizq == 1 && Pd == 2)
    {
        //ocupar palillos
        palillo1 = false;
        palillo2 = false;
        //modificar imagenes
        Invoke((Delegate)new Action(() => {
            //visibilidad de las imagenes

```

Figura 8 Función de parámetros

Para la función que recibe los parámetros y se ejecuta en los hilos principales, donde recibe el número del filósofo que recupero del parámetro en el hilo, este hará el cálculo del palillo izquierdo tanto como el derecho, el palillo derecho será igual al número del filósofo y el palillo izquierdo será igual al número de filosofo más uno ya que se aumenta el número del palillo, esto solamente aplica para el 1,2,3 y 4, para el 5 se condiciona a 1. Ya que no existe un palillo 6.

Aquí se inicializa el semáforo donde el semáforo esta especificado que se inicializa y hace una condición donde se evalúa el palillo izquierdo y derecho pasándole la condición si palillo izquierdo es igual a 1 y palillo derecho es igual a 2 y este posteriormente hace un subprocesso y crea una acción de modificar las imágenes

que no están visibles y pasar las otras imágenes a ocultar así igual con los palillos, dentro de este proceso cambia su estado de ocupación de los palillos.

```
//inicializar semaforo
sem.WaitOne();
if(Pizq == 1 && Pd == 2)
{
    //ocupar palillos
    palillo1 = false;
    palillo2 = false;
    //modificar imagenes
    Invoke((Delegate)new Action(() => {
        //visibilidad de las imagenes
        bob esponja1.Visible = false;
        bob toronja1.Visible = true;
        //visibilidad de los palillos
        palo1.Visible = false;
        palo2.Visible = false;
    }));
    //dormir hilo
    Thread.Sleep(3000);
}
```

Figura 9 Semáforo inicialización

Después de dormir el hilo durante 3 segundos o 3000 microsegundos, se modifica la visibilidad de las imágenes de los avatares para regresar a su estado normal, que es ocultando la imagen de comer y mostrando los palillos de nuevo y dándole un estado verdadero a estos mismo para su ocupación en otro proceso.

```
});
//dormir hilo
Thread.Sleep(3000);
//en otro caso modificar visibilidad de bob esponja
Invoke((Delegate)new Action(() => {
    //visibilidad del avatar
    bob esponja1.Visible = true;
    bob toronja1.Visible = false;
    //visibilidad de los palos
    palo1.Visible = true;
    palo2.Visible = true;
}));
//soltar palillos
palillo2 = true;
palillo3 = true;
```

Figura 10 En otro caso del if

Este proceso se repite para cada uno de los palillos tales son los casos del 2 y 3, 3 y 4, 4 y 5 y por ultimo el caso de 5 y 1.

Por siguiente es necesario verificar el estado de los palillos si es que están disponibles y hacer prácticamente las acciones antes mencionadas, las modificaciones de las imágenes y los estados de los palillos. Para después de dormir soltar los palillos y dejarlos libres

```
if (palillo1 == true && palillo2 == true)
{
    //ocupar palillos
    palillo1 = false;
    palillo2 = false;
    //modificar imagenes
    Invoke((Delegate)new Action(() =>
    {
        //visibilidad de las imagenes
        bobsponja1.Visible = false;
        bobtoronja1.Visible = true;
        //visibilidad de los palillos
        palo1.Visible = false;
        palo2.Visible = false;
    }));
}
```

Figura 11 Verificar el estado de los palillos

La función por último termina el semáforo y lo vuelve a inicializar a su estado inicial, para ocuparlo de nuevo en el proceso que se requiera.

```
        //visibilidad de avatar
        bobsponja5.Visible = true;
        bobtoronja5.Visible = false;
        //visibilidad de palillos
        palo5.Visible = true;
        palo1.Visible = true;
    });
    //desocupar palillos
    palillo5 = true;
    palillo1 = true;
}
//reiniciar semaforo
sem.Release();
}
```

Figura 12 Finalización de semáforo

Pruebas

Para las pruebas del programa es necesario ver como se inicializa el programa, este empieza lanzando la ventana del form donde se encuentran nuestros 5 filósofos y los 5 palillos, estos podrán empezar a comer una vez se haya dado al botón de estoy listo, y empieza la ejecución de los hilos puestos en este.

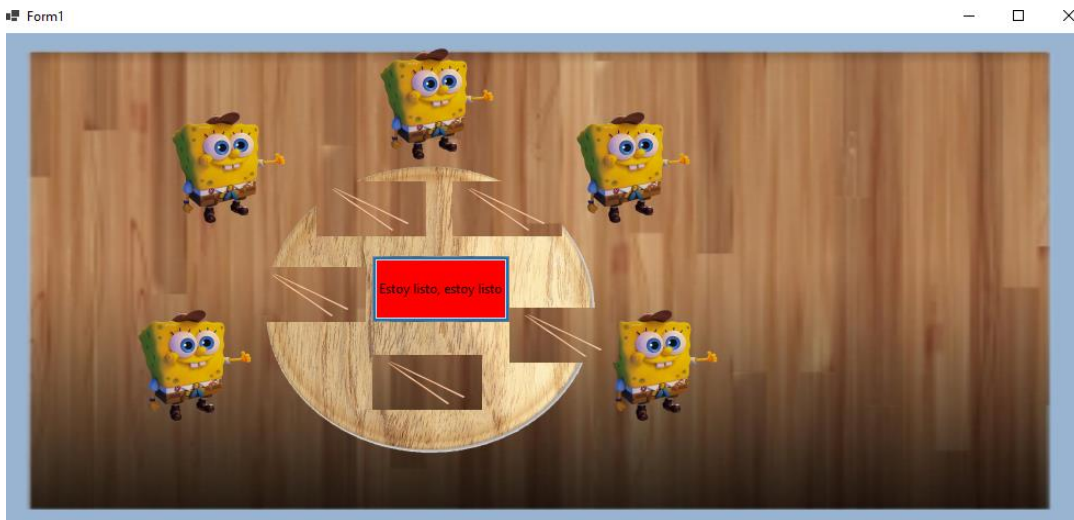


Figura 13 Form de la ventana

En la ejecución del programa se ve como cambia el aspecto de los filósofos cuando estos están comiendo, así como también como desaparecen los utensilios que están ocupando para comer, esto haciendo que solo puedan comer 2 filósofos de los 5 para que no exista un problema dentro de los recursos del programa.

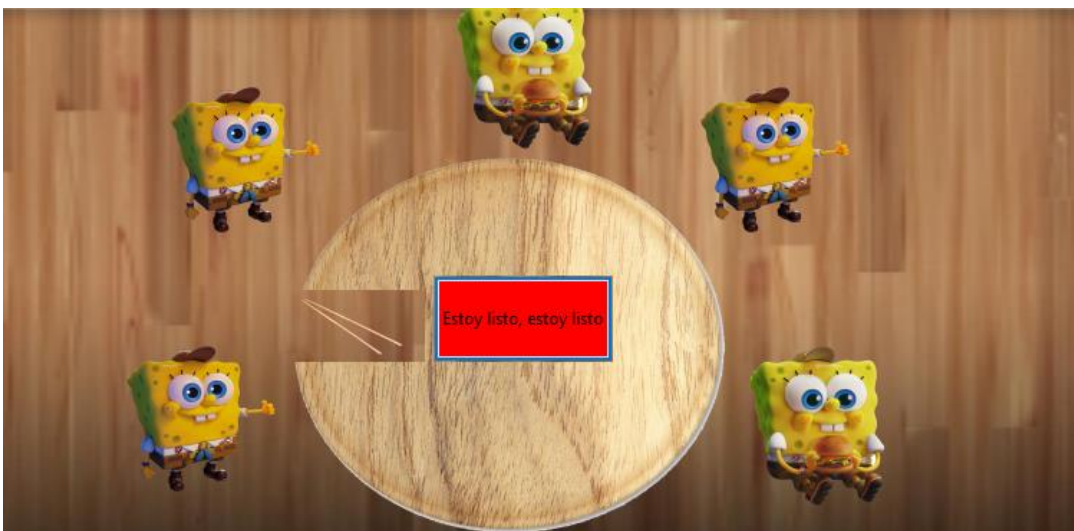


Figura 14 Pruebas de escritorio

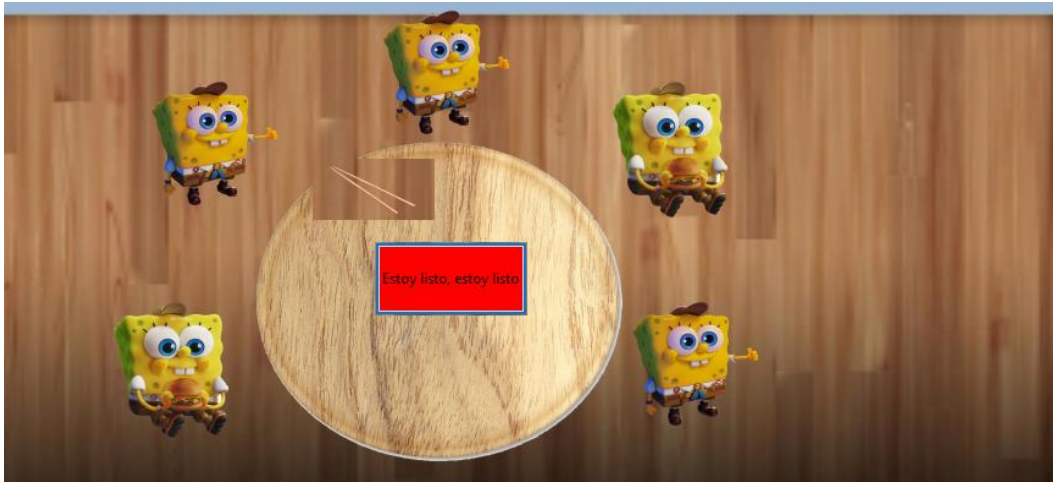
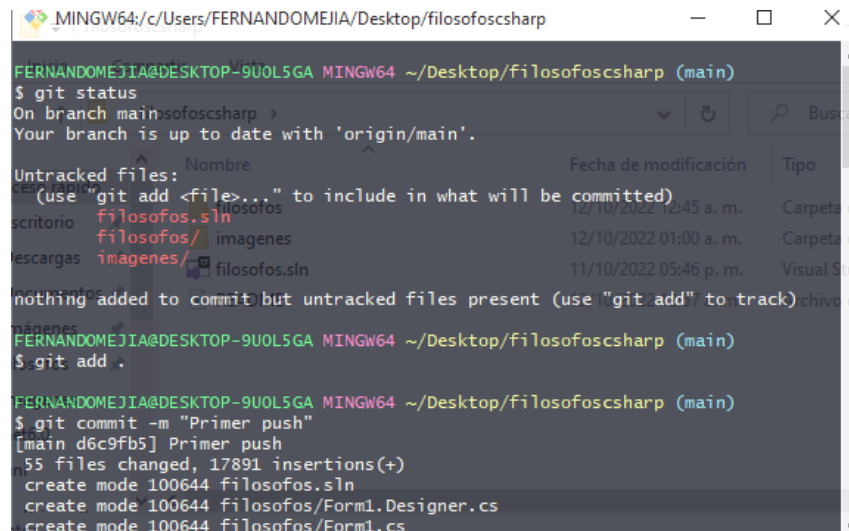


Figura 15 Pruebas de escritorio 2

Subida de GitHub

Para saber el estado de nuestro repositorio es necesario poner el código de git status que nos permitirá acceder al estado actual de versión del repositorio.



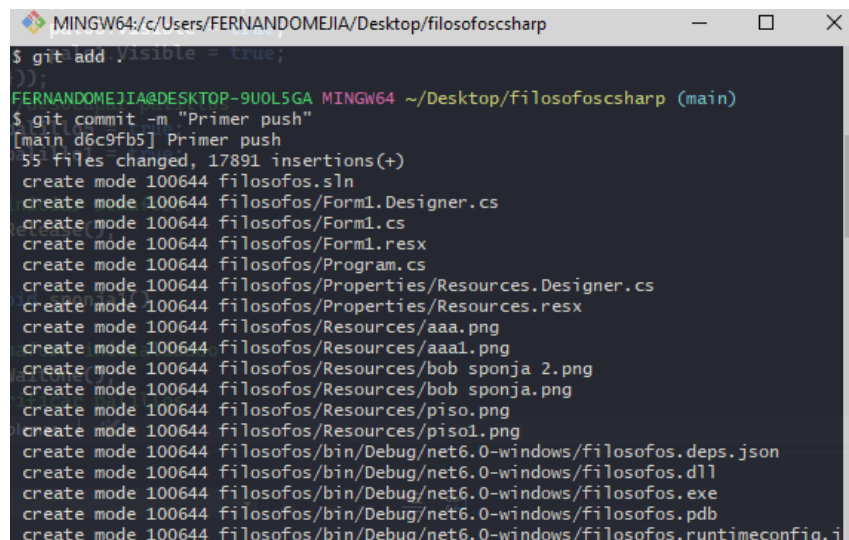
```
MINGW64: c:/Users/FERNANDOMEJIA/Desktop/filosofoscsharp
FERNANDOMEJIA@DESKTOP-9U0L5GA MINGW64 ~/Desktop/filosofoscsharp (main)
$ git status
On branch main:
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    filosofos.sln
    filosofos/imagenes
    filosofos/Form1.Designer.cs

nothing added to commit but untracked files present (use "git add" to track)
FERNANDOMEJIA@DESKTOP-9U0L5GA MINGW64 ~/Desktop/filosofoscsharp (main)
$ git add .
FERNANDOMEJIA@DESKTOP-9U0L5GA MINGW64 ~/Desktop/filosofoscsharp (main)
$ git commit -m "Primer push"
[main d6c9fb5] Primer push
55 files changed, 17891 insertions(+)
create mode 100644 filosofos.sln
create mode 100644 filosofos/Form1.Designer.cs
create mode 100644 filosofos/Form1.cs
```

Figura 16 Git status

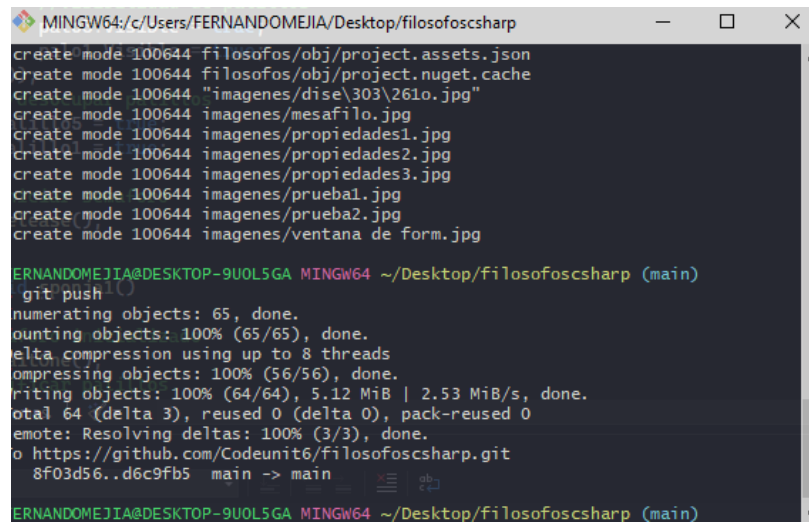
En este comando es necesario para crear la subida de los archivos, así como crear el comentario con el que se van a subir en el repositorio.



```
MINGW64: c:/Users/FERNANDOMEJIA/Desktop/filosofoscsharp
FERNANDOMEJIA@DESKTOP-9U0L5GA MINGW64 ~/Desktop/filosofoscsharp (main)
$ git add .
FERNANDOMEJIA@DESKTOP-9U0L5GA MINGW64 ~/Desktop/filosofoscsharp (main)
$ git commit -m "Primer push"
[main d6c9fb5] Primer push
55 files changed, 17891 insertions(+)
create mode 100644 filosofos.sln
create mode 100644 filosofos/Form1.Designer.cs
create mode 100644 filosofos/Form1.cs
create mode 100644 filosofos/Form1.resx
create mode 100644 filosofos/Program.cs
create mode 100644 filosofos/Properties/Resources.Designer.cs
create mode 100644 filosofos/Properties/Resources.resx
create mode 100644 filosofos/Resources/aaa.png
create mode 100644 filosofos/Resources/aaal.png
create mode 100644 filosofos/Resources/bob sponja 2.png
create mode 100644 filosofos/Resources/bob sponja.png
create mode 100644 filosofos/Resources/piso.png
create mode 100644 filosofos/Resources/piso1.png
create mode 100644 filosofos/bin/Debug/net6.0-windows/filosofos.deps.json
create mode 100644 filosofos/bin/Debug/net6.0-windows/filosofos.dll
create mode 100644 filosofos/bin/Debug/net6.0-windows/filosofos.exe
create mode 100644 filosofos/bin/Debug/net6.0-windows/filosofos.pdb
create mode 100644 filosofos/bin/Debug/net6.0-windows/filosofos.runtimeconfig.json
```

Figura 17 Git add .

El comando git push permite hacer la subida de archivos al repositorio para visualizarlos en el repositorio.

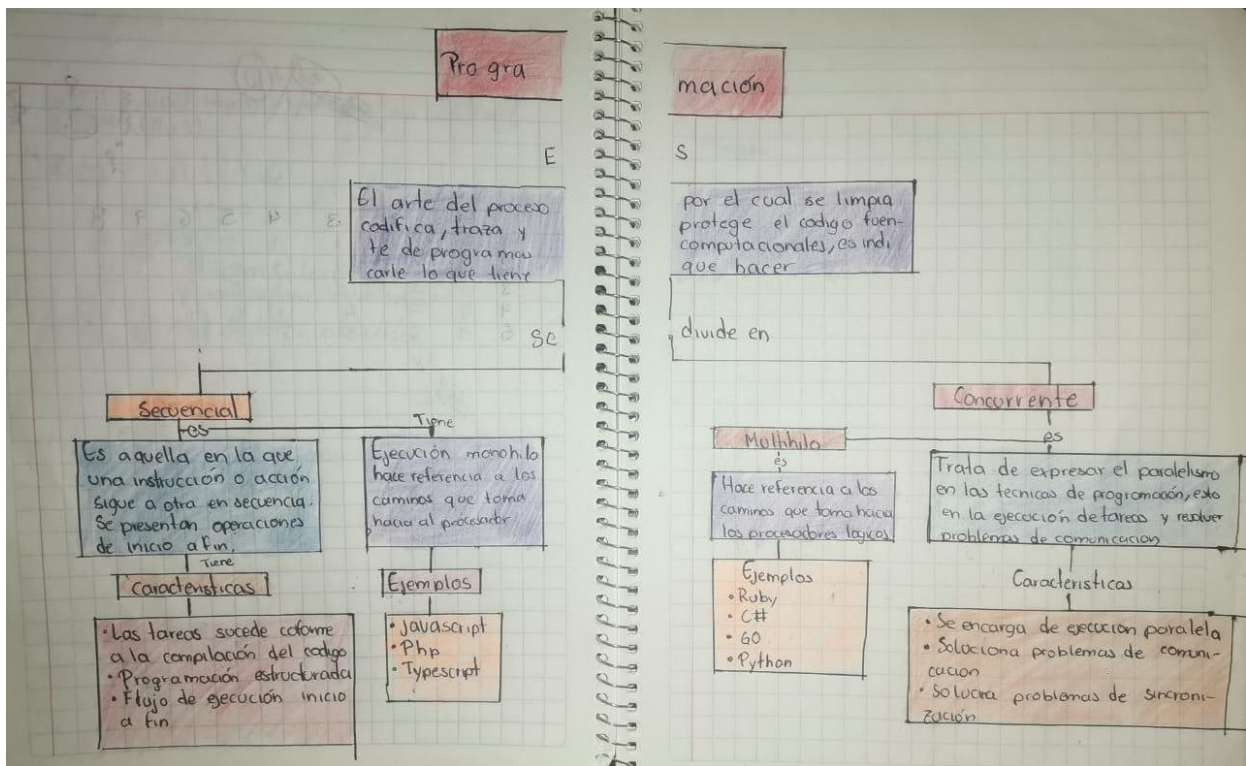
A screenshot of a terminal window titled 'MINGW64: c:/Users/FERNANDOMEJIA/Desktop/filosofoscsharp'. The terminal shows the output of a 'git push' command. It lists several files being created: 'filosofos/obj/project.assets.json', 'filosofos/obj/project.nuget.cache', and several image files in the 'imagenes' directory. The push process is shown with progress for counting, compressing, and writing objects. The final output shows the commit hash '8f03d56..d6c9fb5' and the branch 'main' being pushed to the remote repository 'https://github.com/Codeunit6/filosofoscsharp.git'.

```
MINGW64: c:/Users/FERNANDOMEJIA/Desktop/filosofoscsharp
create mode 100644 filosofos/obj/project.assets.json
create mode 100644 filosofos/obj/project.nuget.cache
create mode 100644 "imagenes/dise\303\261o.jpg"
create mode 100644 imagenes/mesafilo.jpg
create mode 100644 imagenes/propiedades1.jpg
create mode 100644 imagenes/propiedades2.jpg
create mode 100644 imagenes/propiedades3.jpg
create mode 100644 imagenes/prueba1.jpg
create mode 100644 imagenes/prueba2.jpg
create mode 100644 imagenes/ventana de form.jpg



ERNANDOMEJIA@DESKTOP-9U0L5GA MINGW64 ~/Desktop/filosofoscsharp (main)
git push
Enumerating objects: 65, done.
Counting objects: 100% (65/65), done.
Delta compression using up to 8 threads
Compressing objects: 100% (56/56), done.
Writing objects: 100% (64/64), 5.12 MiB | 2.53 MiB/s, done.
Total 64 (delta 3), reused 0 (delta 0), pack-reused 0
Remote: Resolving deltas: 100% (3/3), done.
To https://github.com/Codeunit6/filosofoscsharp.git
 8f03d56..d6c9fb5  main -> main
```

Figura 18 Git push

Mapa conceptual



Rubrica mapa conceptual

		LISTA DE COTEJO MAPA CONCEPTUAL PROGRAMACION CONCURRENTES Y SECUENCIAL UNIDAD 1			
DATOS GENERALES DEL PROCESO DE EVALUACIÓN					
NOMBRE DEL ALUMNO: Agia Gomez Fernando Brayan					
MATRICULA: 1320114054			FECHA: 9 SEPTIEMBRE 2021		
NOMBRE DEL PRODUCTO: MAPA MENTAL PROGRAMACION CONCURRENTES Y SECUENCIAL					
NOMBRE DE LA ASIGNATURA: PROGRAMACIÓN CONCURRENTES			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		100		

[Firma]

Conclusiones

La programación concurrente sin duda ha sido de gran ayuda para algunas de las problemáticas que representan un problema para el consumo de tiempo, gracias a estos se han ahorrado tiempo en la ejecución de estos, pero también representaban un problema que es compartir recursos, hoy en día existen muchas maneras de compartir recursos sin que exista un deadlock o corrupción de datos tales estructuras como lo son monitores, semáforos, mutex y lock. Estas estructuras de carrera permiten que se pueda hacer la ejecución de un programa sin que se corrompa ningún dato esto con el fin de que se tengan datos veraces y exactos al resultado esperado, esto con el fin de reducir el margen de error que estos contienen.

En este documento se abordó la problemática de los filósofos los cuales se tenían que compartir recursos para su funcionamiento y se logro gracias al uso de la estructura semáforo el cual ayudo que este pudiese ejecutarse con éxito sin la corrupción de datos o que se produzca un deadlock, esto con el fin de reforzar el conocimiento y la importancia de evitar la corrupción de datos es importante para problemas similares, esto hace llegar al punto de conclusión que es que la concurrencia es realmente importante para algunos de los problemas que existen y representa la mejor solución, nosotros como futuros ingenieros en software representa la importancia de la utilización de estas tecnologías para futuros proyectos que se realicen, también con el fin de presentar esta evidencia para la asignatura de programación web y poder recalcar nuevamente la importancia de la corrupción de datos.