



# **UNIVERSIDAD POLITECNICA DE TECAMAC**

**LUIS IVÁN TENORIO GARCÍA**

**No. De Control: 1320114181**

**luis\_1320114181@uptecamac.edu.mx**

**7mo. CUATRIMESTRE**

**2722IS**

**INGENIERIA EN SOFTWARE**

**Programación Concurrente**

**PROF: Benito Ramírez Fuentes**

**Septiembre – diciembre 2022**

## Índice

<b>Introducción.....</b>	<b>1</b>
<b>Reporte del Proyecto.....</b>	<b>2</b>
<b>Mapa Conceptual de Diferencias de Programación concurrente y Programación secuencial .....</b>	<b>6</b>
<b>Rúbrica del Mapa Conceptual .....</b>	<b>7</b>
<b>Procedimiento al subir a un repositorio. ....</b>	<b>8</b>

## Tabla de Figuras

Figura 1 Diseño del Form .....	2
Figura 2 Diseño del Form indicando que los Filósofos comen .....	2
Figura 3 Hilos .....	3
Figura 4 Implementación del Semáforo .....	3
Figura 5 Codificación del Semáforo Filósofo 1 .....	3
Figura 6 Codificación del Semáforo Filósofo 2 .....	4
Figura 7 Codificación del Semáforo Filósofo 3 .....	4
Figura 8 Codificación del Semáforo Filósofo 4 .....	4
Figura 9 Codificación del Semáforo Filósofo 5 .....	5
Figura 10 Demostración de ejecución .....	5

## Introducción

La programación concurrente hoy en día es muy importante en la rama de la programación, gracias a que de esta manera de programación nos ayuda a optimizar mucho el tiempo porque se encarga en la ejecución simultanea de múltiples tareas interactivamente y estas tareas pueden ser un conjunto de procesos o hilos de ejecución creados por único programa.

Pero esta manera de programación tiene una serie de problemas muy particulares derivados de las características de la concurrencia, como por ejemplo la violación de la exclusión mutua.

Gracias a la programación concurrente es utilizada a para expresar la concurrencia entre tareas y soluciones de cualquier problema que pudiera surgir de la sincronización y comunicación entre los procesos que participan.

Y es lo que se vera reflejado en este proyecto, se debe de realizar el problema de los “Filósofos Comensales”, este problema consiste en que...

Cinco filósofos están sentados alrededor de una mesa y cada filosofo para poder comer debe de tomar el tenedor que esta a su izquierda y derecha de cada uno de ellos. Si un filosofo toma un tenedor el que esta alado de él no podrá utilizarlo ya que el otro filosofo está ocupándolo. El problema consiste es que deben de comer dos filósofos a la vez pero que no ocupen los mismos tenedores.

Este es el propósito del proyecto, utilizar la programación concurrente para poder resolver el problema de los filósofos.

## Reporte del Proyecto

Este sería el diseño del Form, se visualiza el problema de los filósofos que están sentados alrededor de la mesa esperando comer, de igual se implementó un botón para poner en ejecución el programa.

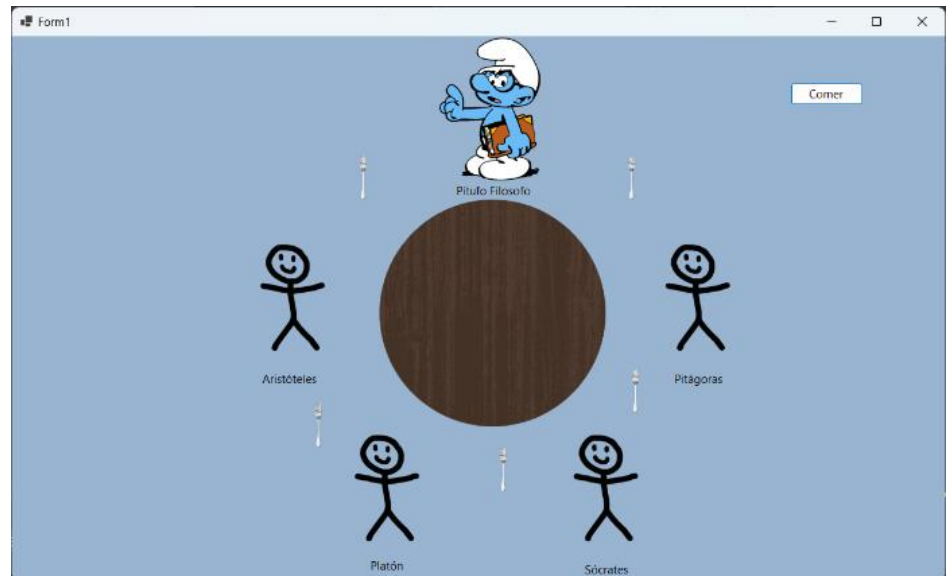
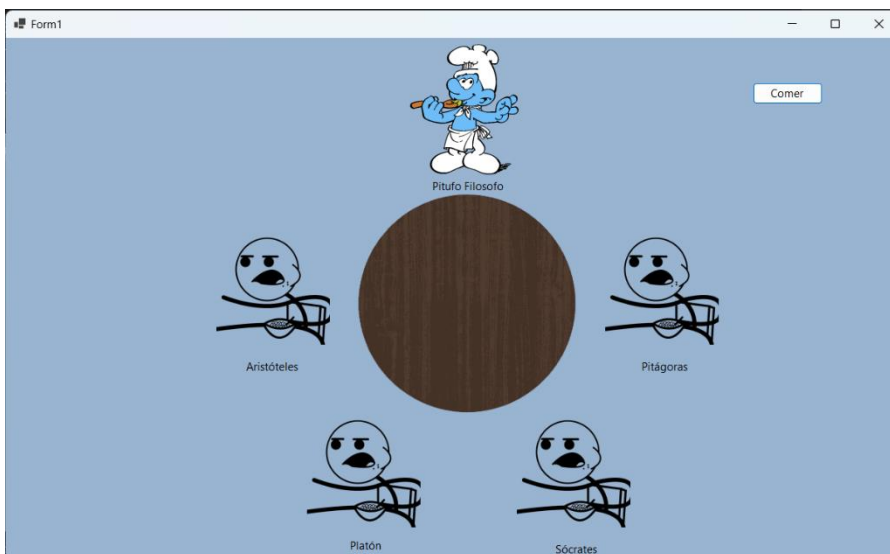


Figura 1 Diseño del Form



Al momento de presionar el botón que dice “comer” las imágenes de los filósofos cambiarán, indicando así que los filósofos están utilizando los tenedores y están comiendo.

Figura 2 Diseño del Form indicando que los Filósofos comen

Esta sería la creación de los hilos, indicando que al momento de dar click en el botón que dice comer se pondrán en funcionamiento los hilos dándole turno a los filósofos de comer.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    //HILOS
    Thread f1 = new Thread(fil1);
    Thread f2 = new Thread(fil2);
    Thread f3 = new Thread(fil3);
    Thread f4 = new Thread(fil4);
    Thread f5 = new Thread(fil5);
    f1.Start();
    f2.Start();
    f3.Start();
    f4.Start();
    f5.Start();
}
```

Figura 3 Hilos

```
3 referencias
public partial class Form1 : Form
{
    //Declaramos los tenedores para despues utilizarlos en el semaforo
    public bool tenedor1 = true,
        tenedor2 = true,
        tenedor3 = true,
        tenedor4 = true,
        tenedor5 = true;
    // Esto significa que dos procesos estan ejecutando y tres estan en espera
    public Semaphore semaforo = new Semaphore(2, 3);
}
```

En esta sección declaramos los tenedores los cuales nos indicaran cuales se están utilizando, de igual forma se realiza el semáforo

Figura 4 Implementación del Semáforo

En esta parte se puede observar lo que se hizo para que el semáforo hiciera su trabajo, indicando que si el filósofo 1 está comiendo agarrara los tenedores 1 y 2 y al momento que haga la acción esos mismos tenedores desaparecerán y cuando el filósofo deje de comer los tenedores regresaran.

```
1 referencia
//if donde indica si el primer filosofo esta comiendo y agarra los tenedores
public void fill()
{
    semaforo.WaitOne(); //Indica que comienza el semaforo
    if(tenedor1 == true && tenedor2 == true) //Se utilizaran tenedor 1 y 2
    {
        tenedor1 = false; //Indica que se utilizara el tenedor 1
        tenedor2 = false; //Indica que se utilizara el tenedor 2
        Invoke((Delegate)new Action() =>
        {
            filosofo1.Visible = false; //Indica que la primer imagen no se visualiza
            come.Visible = true; //La segunda imagen donde indica que esta comiendo se visualiza
            ten1.Visible = false; //Los tenedores 1 y 2 desaparecen
            ten2.Visible = false;
        });
        Thread.Sleep(2000); //Tiempo de espera
    }
    Invoke((Delegate)new Action() =>
    {
        filosofo1.Visible = true; //El filosofo que no come se visualiza
        come.Visible = false; //La imagen del filosofo que come no se visualiza
        ten1.Visible = true; //Los tenedores 1 y 2 se visualizan
        ten2.Visible = true;
    });
    tenedor2 = true;
    tenedor3 = true;
    semaforo.Release(); //se reinicia el semaforo
}
```

Figura 5 Codificación del Semáforo Filósofo 1

Es la misma codificación que con el semáforo 1, pero en este caso el filósofo 2 al momento de comer agarrara el tenedor 2 y 3

```
public void fil2()
{
    semaforo.WaitOne();
    if(tenedor2 == true && tenedor3 == true)//Se utilizara tenedor 2 y 3
    {
        tenedor2 = false;//Indica que se utilizara el tenedor 2
        tenedor3 = false;//Indica que se utilizara el tenedor 3
        Invoke((Delegate)new Action() =>
        {
            filosofo2.Visible = false;
            comer1.Visible = true;
            ten2.Visible=false;
            ten3.Visible = false;
        }));
        Thread.Sleep(2000);
    }
    Invoke((Delegate)new Action() =>
    {
        filosofo2.Visible=true;
        comer1.Visible=false;
        ten2.Visible = true;
        ten3.Visible = true;
    }));
    tenedor3=true;
    tenedor4 = true;
    semaforo.Release();
}
```

Figura 6 Codificación del Semáforo Filósofo 2

```
public void fil3()
{
    semaforo.WaitOne();
    if (tenedor3 == true && tenedor4 == true)//Se utilizaran tenedor 3 y 4
    {
        tenedor3 = false;//Indica que se utilizara el tenedor 3
        tenedor4 = false;//Indica que se utilizara el tenedor 4
        Invoke((Delegate)new Action() =>
        {
            filosofo3.Visible = false;
            comer2.Visible = true;
            ten3.Visible=false;
            ten4.Visible=false;
        }));
        Thread.Sleep(2000);
    }
    Invoke((Delegate)new Action() =>
    {
        filosofo3.Visible = true;
        comer2.Visible=false;
        ten3.Visible = true;
        ten4.Visible = true;
    }));
    tenedor4=true;
    tenedor5=true;
    semaforo.Release();
}
```

Figura 7 Codificación del Semáforo Filósofo 3

En este caso el filósofo 4 utilizara los tenedores 4 y 5

```
public void fil4()
{
    semaforo.WaitOne();
    if (tenedor4 == true && tenedor5 == true){//se utiliza tenedor 4 y 5
        tenedor4 = false;//Indica que se utilizara el tenedor 4
        tenedor5 = false;//Indica que se utilizara el tenedor 5
        Invoke((Delegate)new Action() =>
        {
            filosofo4.Visible = false;
            comer3.Visible = true;
            ten4.Visible = false;
            ten5.Visible = false;
        }));
        Thread.Sleep(2000);
    }
    Invoke((Delegate)new Action() =>
    {
        filosofo4.Visible = true;
        comer3.Visible = false;
        ten4.Visible = true;
        ten5.Visible = true;
    }));
    tenedor5 = true;
    tenedor1 = true;
    semaforo.Release();
}
```

Figura 8 Codificación del Semáforo Filósofo 4

```
public void fil5()
{
    semaforo.WaitOne();
    if (tenedor5 == true && tenedor1 == true)//Se utilizaran tenedor 5 y 1
    {
        tenedor5 = false;//Indica que se utilizara el tenedor 5
        tenedor1 = false;//Indica que se utilizara el tenedor 1
        Invoke((Delegate)new Action() =>
        {
            filosofo5.Visible = false;
            comer4.Visible = true;
            ten5.Visible = false;
            ten1.Visible = false;
        });
        Thread.Sleep(2000);
    }
    Invoke((Delegate)new Action() =>
    {
        filosofo5.Visible = true;
        comer4.Visible = false;
        ten5.Visible = true;
        ten1.Visible = true;
    });
    tenedor5 = true;
    tenedor1 = true;
    semaforo.Release();
}
```

Ya como es el ultimo filosofo este utilizara el tenedor numero 5 y regresara al tenedor 1, ya que se completó una vuelta completa.

Figura 9 Codificación del Semáforo Filósofo 5

Al momento de correr el programa se visualizará así, mostrara los filósofos que están comiendo y los tenedores que están ocupando.

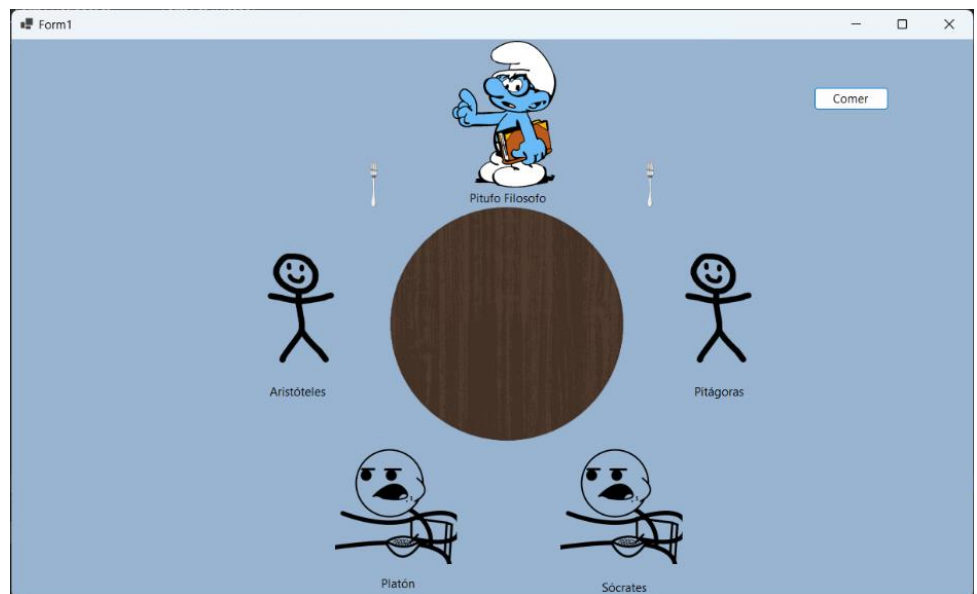
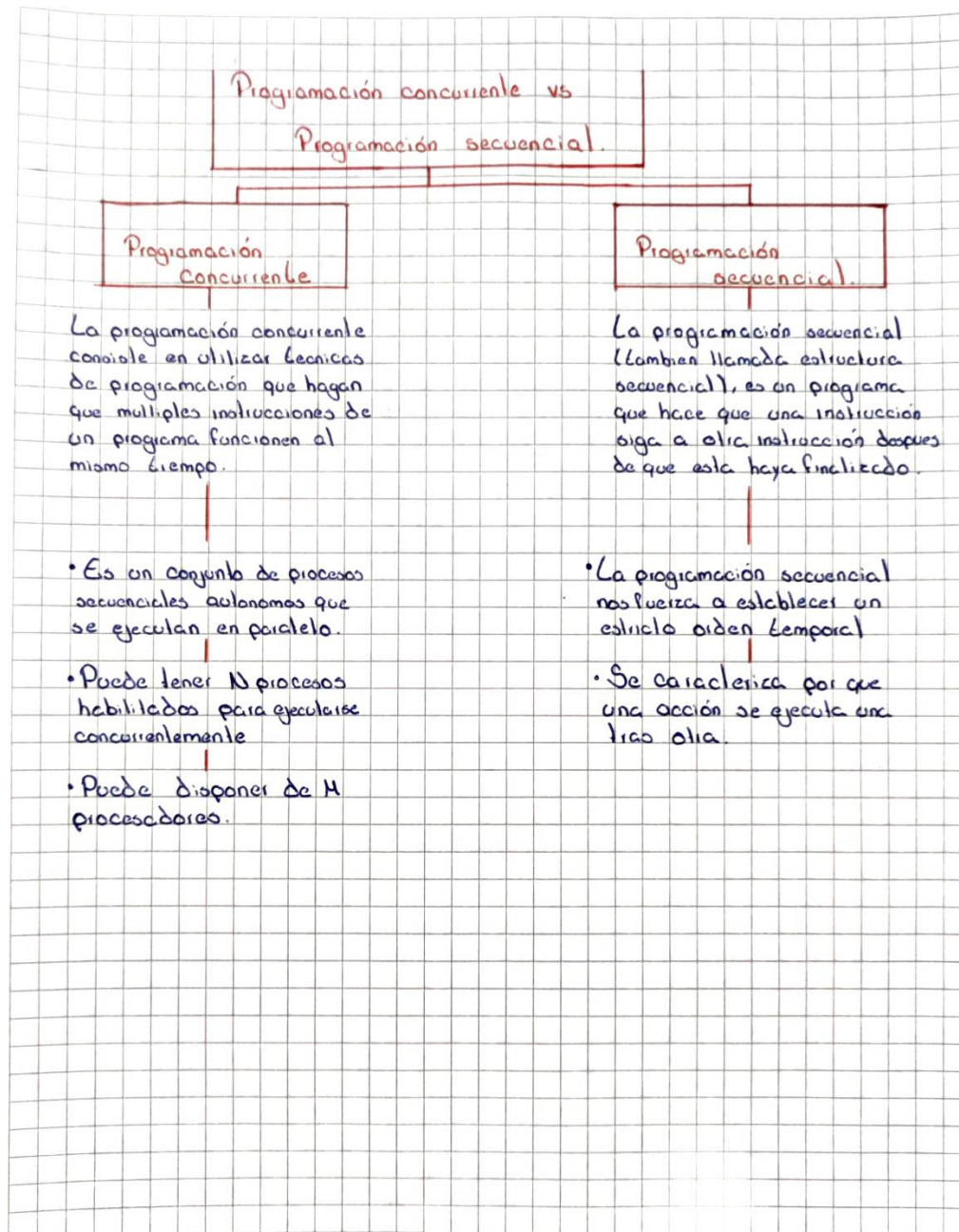


Figura 10 Demostración de ejecución



## Mapa Conceptual de Diferencias de Programación concurrente y Programación secuencial



5 / 10 / 22



Scribe



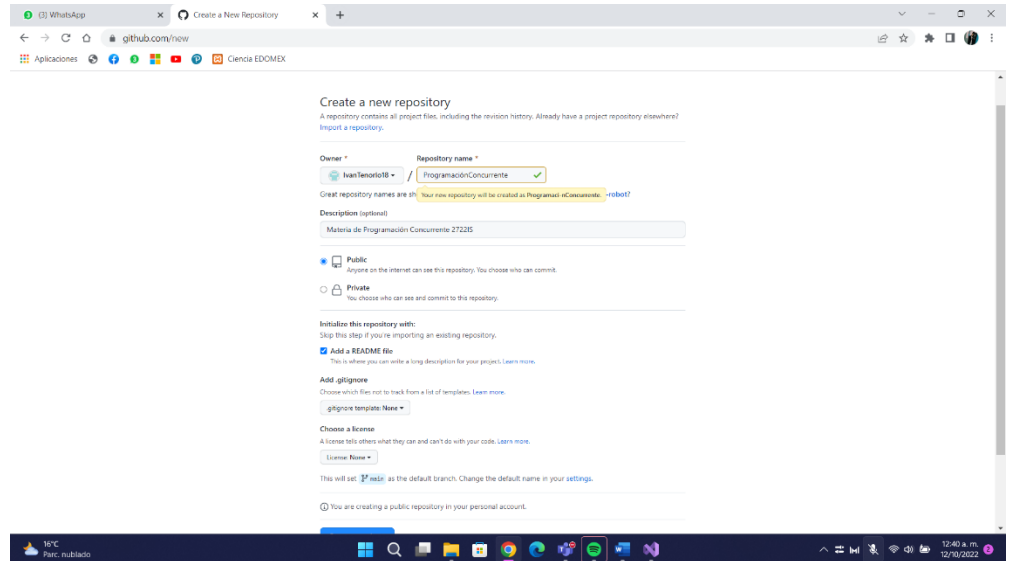
## Rúbrica del Mapa Conceptual

 <b>LISTA DE COTEJO MAPA CONCEPTUAL PROGRAMACION CONCURRENTRE Y SECUENCIAL</b> <b>UNIDAD 1</b> 			
<b>DATOS GENERALES DEL PROCESO DE EVALUACIÓN</b>			
NOMBRE DEL ALUMNO: <u>Tenorio Garcia Luis Iván</u>			
MATRICULA: <u>1320114181</u>	FECHA: <u>9 SEPTIEMBRE 2021</u>		
NOMBRE DEL PRODUCTO: <u>MAPA MENTAL PROGRAMACION CONCURRENTRE Y SECUENCIAL</u>			
NOMBRE DE LA ASIGNATURA: <u>PROGRAMACIÓN CONCURRENTRE</u>	CUATRIMESTRE O CICLO DE FORMACIÓN: <u>SÉPTIMO CUATRIMESTRE</u>		
NOMBRE DEL DOCENTE: <u>BENITO RAMIREZ FUENTES</u>			
Instrucciones: Realizar un mapa conceptual sobre las diferencias entre la programación concurrente y la secuencial implementación semáforo			
<b>INSTRUCCIONES</b>			
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	<u>100</u>

*[Firma]*

## Procedimiento al subir a un repositorio.

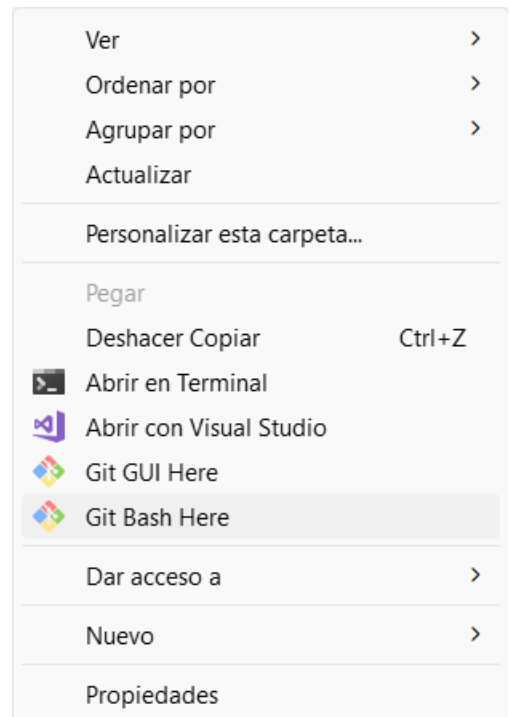
Como primer paso debemos de crear un repositorio, en este caso en GitHub, en este caso se llamara como la materia “Programación Concurrente”



Después clonamos el repositorio en una carpeta con el siguiente comando “git clone -aquí ponemos el link del repositorio”

Después de que hayamos clonado el repositorio pasamos la carpeta del proyecto en el repositorio que clonamos, para poderlo subir debemos de seguir los siguientes comandos.

Ya que este la carpeta del proyecto le damos clic derecho a la raíz de la carpeta clonada y le damos en “Git Bash Here”



Y se nos abrirá una consola, en esa consola escribimos el siguiente comando “git status” y nos aparecerá el nombre de la carpeta del proyecto en rojo.

```
MINGW64:/c:/Users/ivan_/OneDrive/Documentos/Programaci-nConcurrente
ivan_@LAPTOP-0J3H972A MINGW64 ~/OneDrive/Documentos/Programaci-nConcurrente (mai
n)
$ 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)
  Filofofos/

nothing added to commit but untracked files present (use "git add" to track)
```

```
ivan_@LAPTOP-0J3H972A MINGW64 ~/OneDrive/Documentos/Programaci-nConcurrente (mai
n)
$ git add .
```

El siguiente código que pondremos será “git add .” esto es para que se suba la carpeta al repositorio.

Posterior mente ponemos el comando de “git commit -m” Cualquier Nombre” esto nos ayudara a ponerle nombre al proyecto al momento de subirlo.

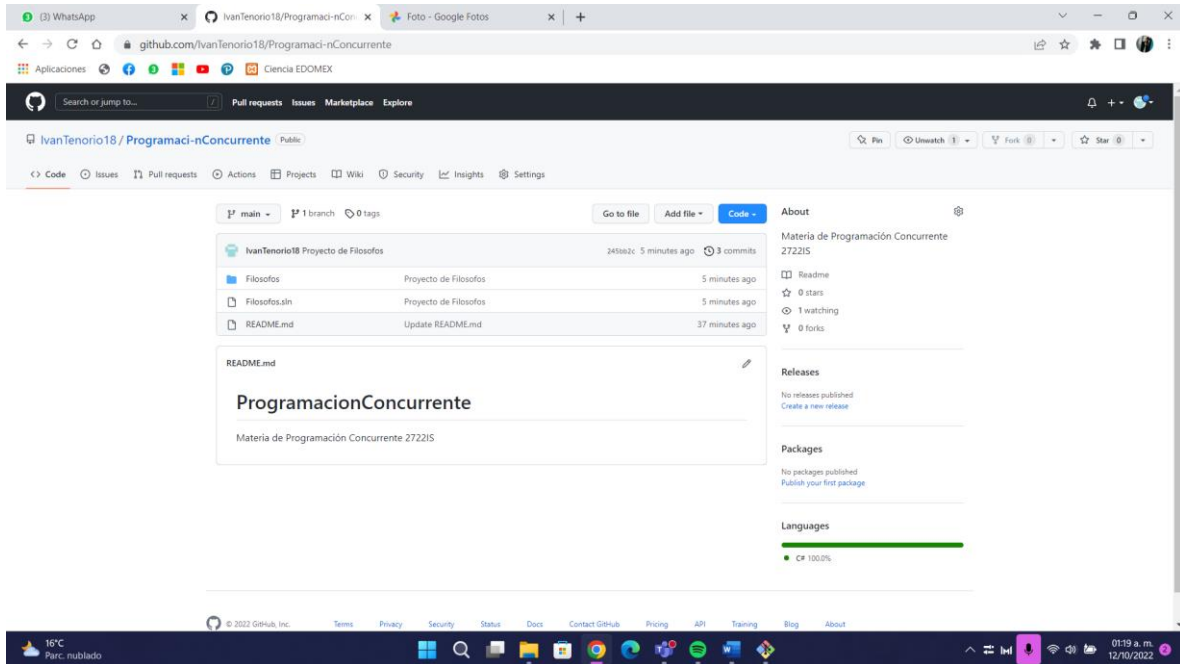
Después de ese comando se pone el siguiente comando “git push” esto es para que se suban todos los archivos correctamente.

```
MINGW64:/c:/Users/ivan_/OneDrive/Documentos/Programaci-nConcurrente
create mode 100644 Filofofos/obj/Debug/net6.0-windows/ref/Filofofos.dll
create mode 100644 Filofofos/obj/Debug/net6.0-windows/refint/Filofofos.dll
create mode 100644 Filofofos/obj/Filofofos.csproj.nuget.dgspec.json
create mode 100644 Filofofos/obj/Filofofos.csproj.nuget.g.props
create mode 100644 Filofofos/obj/Filofofos.csproj.nuget.g.targets
create mode 100644 Filofofos/obj/project.assets.json
create mode 100644 Filofofos/obj/project.nuget.cache

ivan_@LAPTOP-0J3H972A MINGW64 ~/OneDrive/Documentos/Programaci-nConcurrente (mai
n)
$ git push
Enumerating objects: 57, done.
Counting objects: 100% (57/57), done.
Delta compression using up to 8 threads
Compressing objects: 100% (48/48), done.
Writing objects: 100% (56/56), 992.38 KiB | 734.00 KiB/s, done.
Total 56 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/IvanTenorio18/Programaci-nConcurrente.git
16a404b..245bb2c main -> main

ivan_@LAPTOP-0J3H972A MINGW64 ~/OneDrive/Documentos/Programaci-nConcurrente (mai
n)
$
```

Y ya así se subieron los documentos correctamente al repositorio, como se muestra en la imagen y así tendríamos nuestro proyecto en un repositorio.



## Conclusión

Gracias a este trabajo se pudo observar lo importante que es la programación concurrente, que dependiendo el trabajo que soliciten es si nos conviene usar la programación concurrente, pero gracias a ello se nos puede facilitar el trabajo ya que nos acortan tiempo y líneas de código, como se pudo ver reflejado en este proyecto de los filósofos.

Se pudo resolver la problemática de los filósofos mediante la programación concurrente, es decir se ejecutaron múltiples tareas a la misma vez y no de forma secuencial.