Clase 18
Hilos

Programación y Laboratorio II

¿Qué es CONCURRENCIA?

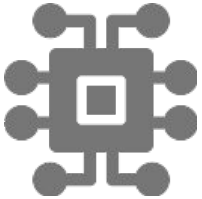
En programación hablamos de **concurrencia** cuando se ejecuta más de una tarea al mismo tiempo.

Esta habilidad es útil cuando necesitamos que una aplicación haga alguna cosa mientras está trabajando en algo más.

Uso de concurrencia en aplicaciones

Permite que:

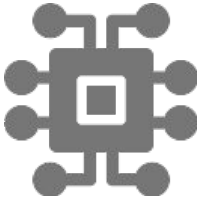
- Los usuarios finales puedan interactuar con la interfaz de usuario de manera no-bloqueante.
- Un servidor pueda atender varias peticiones en simultáneo y no afectar los tiempos de respuesta ante períodos de alta demanda.
- Realizar tareas de cómputo complejas de manera más rápida y haciendo un uso más eficaz los recursos de la computadora.



¿Qué es un HILO?

Un **hilo (thread)**, también llamado **hebra** o **subproceso**, es la unidad básica a la que un sistema operativo asigna tiempo de procesamiento.

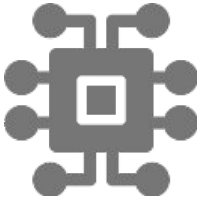
Son los encargados de ejecutar nuestro código sentencia a sentencia.



¿Qué es un PROCESO?

Un **proceso** es un programa en ejecución que tiene asignados recursos tales como memoria e hilos.

Todos los hilos de un mismo proceso comparten los mismos recursos asignados por el sistema operativo.



¿Qué es la PROGRAMACIÓN MULTIHILO?

Por defecto, cada proceso tiene un único hilo.

La **programación multihilo (multithreaded programming)** permite que un proceso se ejecute sobre múltiples hilos y cada uno de esos hilos esté realizando una tarea distinta en paralelo.

*La **programación en paralelo (parallel programming)** es un subtipo de programación multihilo.*
Se utiliza para dividir una gran carga de trabajo en partes independientes y ejecutarlas en paralelo, maximizando el uso de los núcleos de la CPU.

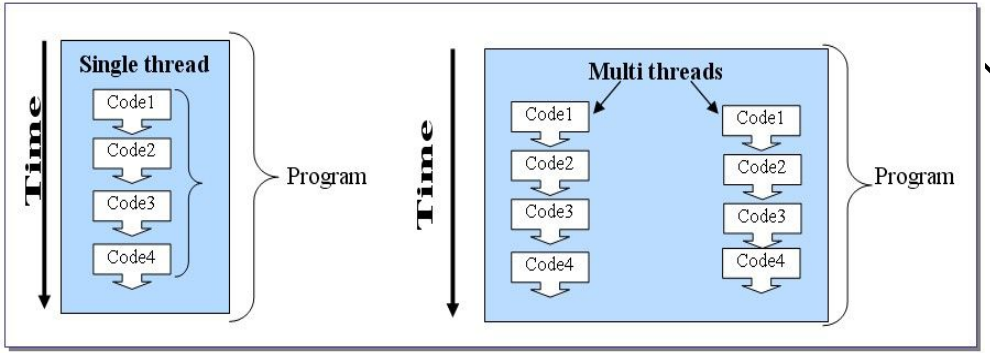
Ciclo de vida de un hilo

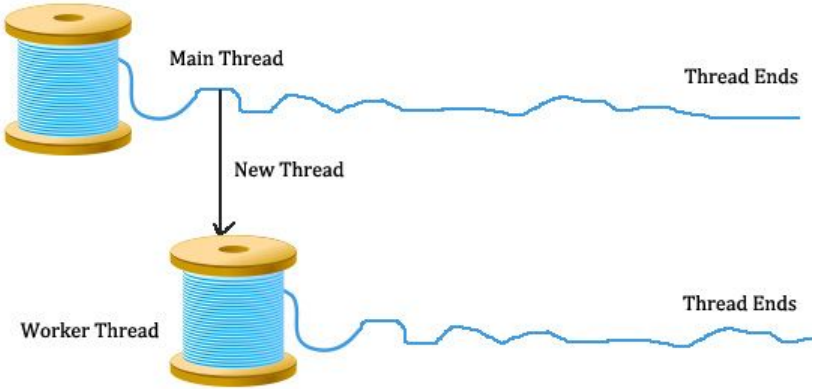
Se llama **hilo principal (main thread)** al primer hilo sobre el que se empezó a ejecutar la aplicación.

A partir de un hilo se pueden abrir nuevos **hilos secundarios**.
Un hilo secundario va a tener su propia **pila de ejecución**, independiente de la de origen.

El proceso existirá mientras al menos uno de sus hilos de ejecución siga activo.

Cuando **todos** los hilos de ejecución finalizan, el proceso no existe más y todos sus recursos son liberados.





Task

- Task es la clase que utilizaremos para ejecutar métodos en un nuevo hilo.
- Task nos permitirá manejar también el estado de dicho hilo.
- Task es la evolución de la clase ***Thread*** de *.Net Framework*.

```
//Agrego las bibliotecas
using System.Threading;
using System.Threading.Tasks;
```

```
//Metodo a ejecutarse en un hilo secundario
1 referencia
static void MiMetodo()
{
    for (int i = 0; i < 100; i++)
    {
        Console.WriteLine($"iteracion numero {i}");
        Thread.Sleep(1000);
    }
}
```

```
0 referencias
static void Main(string[] args)
{
    //Creo la tarea que va ejecutar un metodo en otro hilo
    Task tarea = new Task(MiMetodo);
    //inicio la tarea/hilo
    tarea.Start();
    Console.WriteLine("La tarea esta iniciada e imprimo en paralelo 1");
    Console.WriteLine("La tarea esta iniciada e imprimo en paralelo 2");
    Console.WriteLine("La tarea esta iniciada e imprimo en paralelo 3");
    Console.WriteLine("La tarea esta iniciada e imprimo en paralelo 4");
    //si fuera necesario, wait bloquea el hilo hasta que finalice la tarea
    tarea.Wait();

    Console.WriteLine("Termino la tarea secundaria");

    Console.ReadKey();
}
```

¿En qué hilo me encuentro?

- La propiedad ***Thread.CurrentThread.ManagedThreadId*** retorna el id del hilo en el que se está ejecutando esa instrucción.
- La propiedad ***Task.CurrentId*** retorna el id de la tarea, es decir, de la instancia de *Task*.
- Siguiendo con el ejemplo anterior, podríamos:

```
//Creo la tarea que va ejecutar un metodo en otro hilo
Task tarea = new Task(MiMetodo);
Task tarea2 = new Task(MiMetodo);
Task tarea3 = new Task(MiMetodo);
//inicio la tarea/hilo
tarea.Start();
tarea2.Start();
tarea3.Start();

Console.WriteLine($"id hilo principal {Thread.CurrentThread.ManagedThreadId}");
Console.WriteLine($"id de instancia tarea  {tarea.Id}");
Console.WriteLine($"id de instancia tarea  {tarea2.Id}");
Console.WriteLine($"id de instancia tarea  {tarea3.Id}");
```

```
//Metodo a ejecutarse en un hilo secundario
3 referencias
static void MiMetodo()
{
    Console.WriteLine($"id de la instancia de task en metodo {Task.CurrentId}");
    Console.WriteLine($"Id del hilo donse se esta" +
        $" ejecutando {Thread.CurrentThread.ManagedThreadId}");
    for (int i = 0; i < 20; i++)
    {
        Console.WriteLine($"iteracion numero {i}");
        Thread.Sleep(1000);
    }

}
```

Hilos de Ejecución

- Otra forma de ejecutarlo es mediante el método estático ***Run***.
- Si usamos este método, la tarea se instanciará e inicializará al mismo tiempo, ejecutando el método en algún hilo disponible en el *thread pool*.

```
// Creo el hilo que ejecutará UnMetodo en otro hilo.
Task tarea = Task.Run(UnMetodo);
```

Hilos de Ejecución

- O utilizando expresiones ***lambda*** para crear un método anónimo.
- Esto solo es recomendable si la tarea no se utilizará en otro lugar y es una tarea simple.

```
Task tarea = Task(() => { Thread.Sleep(2000); });
```

Hilos con métodos parametrizados

- El método utilizado puede tener parámetros.
- Para esto deberemos utilizar expresiones Lambda.

```
Task(()=> MetodoConArgumentos(param));
```

Cancelar Hilo

- Debemos instanciar un ***CancellationTokenSource***.
- Obtener de la instanciar un *CancellationToken* con la propiedad **Token**.
- Pasar el *CancellationToken* como argumento a nuestro Task.
- Accionar el *CancellationToken* con el método **Cancel()** cuando se quiera cancelar el hilo.

Cancelar Hilo

```
0 referencias
static void Main(string[] args)
{
    CancellationTokenSource cancellation = new CancellationTokenSource();
    CancellationToken cancellationToken = cancellation.Token;
    Task.Run(() => ImprimirHora(cancellation), cancellationToken);
    Console.WriteLine("Leo teclas");
    int numero = 0;
    ConsoleKey teclaPresionada;
    do
    {
        numero++;
        Console.WriteLine($"Contando al presionar una tecla {numero}");
        teclaPresionada = Console.ReadKey().Key;
        Console.WriteLine($"La tecla presionada es {teclaPresionada}");
    } while (teclaPresionada != ConsoleKey.Escape);

    cancellation.Cancel();


}
```

```
1 referencia
static void ImprimirHora(CancellationTokenSource cancellation)
{

    do
    {
        Console.WriteLine($"La hora actual es: {DateTime.Now}");
        Thread.Sleep(1000);
    } while (!cancellation.IsCancellationRequested);
}
```

Hilos y Controles Visuales

- Si se desea modificar un *control visual* de un formulario (TextBox, ComboBox, Label, etc.) desde un hilo diferente al principal (“dueño” de estos controles) se **debe** invocar a dicho hilo.
- Para esto se le consulta al control si necesita ser invocado el hilo principal (***InvokeRequired***).
- De ser así, se invoca a dicho hilo (***BeginInvoke***) mediante un delegado.

Hilos y Controles Visuales

- Dicha invocación puede necesitar parámetros.
- Para resolver esto, se utiliza un array de *Object*.
- Al realizar el Invoke (sincrónico, espera que un *Thread* finalice para ejecutar otro) o *BeginInvoke* (asincrónico) se pasará el delegado y dicho array.

Ejemplo con controles de Form

```
delegate void Delegado();
2 referencias
private void ActualizarHora()
{
    if (this.InvokeRequired)
    {
        Delegado callback = new Delegado(ActualizarHora);
        this.BeginInvoke(callback);
    }
    else
    {
        this.lblHora.Text = $"Hora: {DateTime.Now}";
    }

}
```

```
1 referencia
private void IniciarReloj
    (CancellationToken cancellation)
{
    do
    {
        this.ActualizarHora();
        Thread.Sleep(1000);
    } while (!cancellation.IsCancellationRequested);

}
```

Ejemplo con controles de Form

```
2 referencias
private void ActualizarHora(DateTime hora)
{
    if (this.InvokeRequired)
    {
        Action<DateTime> callback = new Action<DateTime>(ActualizarHora);
        object[] args = { hora };
        this.BeginInvoke(callback, args);
    }
    else
    {
        this.lblHora.Text = $"Hora: {hora}";
    }

}
```

```
1 referencia
private void IniciarReloj
    (CancellationToken cancellation)
{
    do
    {
        this.ActualizarHora(DateTime.Now);
        Thread.Sleep(1000);
    } while (!cancellation.IsCancellationRequested);

}
```

EJERCICIOS

Realizar los siguiente ejercicios:

- I01 - El relojero
- I02 - Simulador de atención al cliente