

ChatGPT

image generator

Explore GPTs

Today

Ejercicio programación paralela

Yesterday

Random

Previous 30 Days

PC questions

Juego dinosaurio C#

Grammar Correction Assistance

Programacion

Translate

January

Be better

Learning better

2024

English

Incrementar ahorros y trabajo

Learn English Better!

“Eres un profesor experto en programación paralela en C#. Necesito que me generes un ejercicio práctico que cumpla con estos requisitos:

Debe ser un problema del mundo real que requiera procesamiento paralelo

El ejercicio debe obligatoriamente usar estos conceptos: Task.Run y Task.Factory.StartNew

Tareas padre-hijo (TaskCreationOptions.AttachedToParent)

ContinueWith con

Task.ContinuationOptions.OnlyOnRunToCompletion

Task.ContinuationOptions.OnlyOnCanceled para manejo de errores

Task.WhenAny para coordinación de tareas

Task.Delay para simular procesamiento

Tu respuesta debe incluir:

Descripción detallada del problema a resolver

Lista de requisitos específicos

Un ejemplo básico de la estructura esperada del código

Criterios de evaluación

Pistas o sugerencias para la implementación

Ritos adicionales opcionales para estudiantes avanzados

Por favor, asegúrate que el ejercicio sea realizable en 2-3 horas y que tenga un nivel intermedio de dificultad.”

Razonado sobre ejercicio de programación paralela for 13 seconds

A continuación, se presenta un ejercicio práctico que cumple con los requisitos solicitados:

## Ejercicio: Procesamiento Paralelo de Pedidos en una Tienda Online

### Descripción del problema

Imagina que trabajas en el desarrollo del sistema de procesamiento de pedidos de una tienda online. Cada pedido recibido debe pasar por varias etapas: verificación de stock, procesamiento de pago y preparación de envío. Estas etapas se ejecutarán en paralelo para optimizar el tiempo de respuesta. Además, se desea detectar cuál pedido se completa primero y, en caso de error en alguna etapa, manejarlo adecuadamente.

El ejercicio consiste en:

- Simular el procesamiento de múltiples pedidos en forma concurrente.
- Dividir cada pedido en tareas hijo (usando tareas padre-hijo) para cada etapa del procesamiento.
- Usar continuaciones para ejecutar acciones posteriores únicamente cuando las tareas se completan exitosamente, y otra para manejar posibles errores.
- Coordinar el procesamiento mediante Task.WhenAny para identificar el primer pedido completado.
- Emplear Task.Delay para simular tiempos de procesamiento en cada tarea.

### Lista de requisitos específicos

- Simulación real: Procesamiento paralelo de pedidos en una tienda online.
- Inicio de tareas: Utilizar Task.Run y Task.Factory.StartNew para iniciar tareas.
- Tareas padre-hijo: Crear tareas hijo con TaskCreationOptions.AttachedToParent dentro del procesamiento de cada pedido.
- Continuaciones exitosas: Usar ContinueWith con TaskContinuationOptions.OnlyOnRunToCompletion para acciones que se ejecuten solo si las tareas se completan correctamente.
- Manejo de errores: Implementar continuaciones con TaskContinuationOptions.OnlyOnCanceled para detectar y gestionar errores en las tareas.
- Coordinación de tareas: Emplear Task.WhenAny para identificar cuál pedido se procesa primero.
- Simulación de procesamiento: Utilizar Task.Delay para simular tiempos de procesamiento.
- Tiempo estimado: El ejercicio debe poder completarse en 2-3 horas y tener un nivel intermedio de dificultad.

### Ejemplo básico de la estructura esperada del código

```
csharp Copy Edit

using System;
using System.Collections.Generic;
using System.Threading.Tasks;

class Program
{
    static async Task Main(string[] args)
    {
        // Lista de tareas para simular el procesamiento de varios pedidos
        List<Task> pedidos = new List<Task>();

        // Creación y lanzamiento de 5 pedidos
        for (int i = 1; i <= 5; i++)
        {
            int pedidoId = i;
            // Usamos Task.Run para iniciar el procesamiento de cada pedido
            var pedidoTask = Task.Run(() => ProcesarPedido(pedidoId));
            pedidos.Add(pedidoTask);
        }

        // Uso de Task.WhenAny para detectar cuál pedido se procesa primero
        Task primerPedidoCompleto = await Task.WhenAny(pedidos);
        Console.WriteLine("El primer pedido procesado fue el número: " + pedidos.IndexOf(primerPedidoCompleto));

        // Esperar a que todos los pedidos finalicen
        await Task.WhenAll(pedidos);
        Console.WriteLine("Todos los pedidos han sido procesados.");
    }

    static Task ProcesarPedido(int pedidoId)
    {
        // Tarea padre que abarca el procesamiento completo del pedido
        return Task.Factory.StartNew(() =>
        {
            Console.WriteLine($"Iniciando procesamiento del pedido {pedidoId}");

            // Tareas hijo para cada etapa del procesamiento, adjuntas a la tarea padre
            var verificacionStock = Task.Factory.StartNew(async () => {
                await Task.Delay(500); // Simula verificación de stock
                Console.WriteLine($"Pedido {pedidoId}: Stock verificado.");
            }, TaskCreationOptions.AttachedToParent).Unwrap();

            var procesamientoPago = Task.Factory.StartNew(async () => {
                await Task.Delay(700); // Simula procesamiento del pago
                Console.WriteLine($"Pedido {pedidoId}: Pago procesado.");
            }, TaskCreationOptions.AttachedToParent).Unwrap();

            var preparacionEnvio = Task.Factory.StartNew(async () => {
                await Task.Delay(600); // Simula preparación del envío
                Console.WriteLine($"Pedido {pedidoId}: Envío preparado.");
            }, TaskCreationOptions.AttachedToParent).Unwrap();

            // Continuación que se ejecuta únicamente si todas las tareas hijo se completan
            Task finalizacion = Task.WhenAll(verificacionStock, procesamientoPago, preparacionEnvio).ContinueWith(t =>
            {
                Console.WriteLine($"Pedido {pedidoId}: Procesamiento completado exitosamente.");
                TaskContinuationOptions.OnlyOnRunToCompletion;
            });
        }, TaskCreationOptions.AttachedToParent).Unwrap();
    }
}
```

```
finalizacion.ContinueWith(t =>
{
    Console.WriteLine($"Pedido {pedidoId}: Error en el procesamiento.");
}, TaskContinuationOptions.OnlyOnCanceled);

// Esperamos la finalización de las tareas para mantener la sincronía en este eje
finalizacion.Wait();
});
}
}
```

#### Criterios de evaluación

- **Paralelismo:** Uso correcto de `Task.Run` y `Task.Factory.StartNew` para iniciar tareas.
- **Tareas padre-hijo:** Implementación adecuada de tareas adjuntas mediante `TaskCreationOptions.AttachedToParent`.
- **Continuaciones:** Uso correcto de `ContinueWith` con `TaskContinuationOptions.OnlyOnTaskCompletion` y `TaskContinuationOptions.OnlyOnCanceled`.
- **Coordinación:** Empleo de `Task.WhenAny` para identificar el primer pedido completado.
- **Simulación:** Uso de `Task.Delay` para simular tiempos de procesamiento realistas.
- **Legibilidad:** Código bien estructurado, con comentarios claros y siguiendo buenas prácticas de programación.

#### Pistas y sugerencias para la implementación

- **Comprender las diferencias:** Revisa la diferencia entre `Task.Run` y `Task.Factory.StartNew`, y en qué situaciones es recomendable usar cada uno.
- **Uso de continuaciones:** Investiga cómo funcionan las opciones de continuación (`OnlyOnTaskCompletion` y `OnlyOnCanceled`) para asegurarte de que se ejecuten en los escenarios esperados.
- **Adjuntar tareas:** Presta atención a cómo se adjuntan tareas hijo al padre utilizando `TaskCreationOptions.AttachedToParent`, lo cual garantiza que la tarea padre espere la finalización de todas sus tareas hijo.
- **Simulación de tiempo:** Ajusta los valores de `Task.Delay` para simular distintos tiempos de procesamiento y observar el comportamiento en la ejecución paralela.
- **Coordinación de tareas:** Experimenta con `Task.WhenAny` para ver cómo se puede detectar y notificar la finalización del primer pedido procesado.

#### Retos adicionales opcionales para estudiantes avanzados

1. **Reintentos:** Modifica el ejercicio para que, si alguna etapa del pedido falla, se reintente automáticamente hasta 3 veces antes de marcar el pedido como fallido.
2. **Cancelación:** Implementa un mecanismo de cancelación utilizando `CancellationToken` que permita detener el procesamiento de pedidos si se supera un tiempo máximo de espera.
3. **Etapas adicionales:** Añade nuevas etapas al procesamiento (por ejemplo, validación de la dirección de envío, generación de notificaciones al cliente) y coordina sus dependencias.
4. **Medición de rendimiento:** Incorpora un sistema para medir y reportar el tiempo total de procesamiento de cada pedido y del sistema en general, para analizar la eficiencia del procesamiento paralelo.

Este ejercicio brinda la oportunidad de profundizar en los conceptos de programación paralela en C# y es ideal para ser completado en un lapso de 2 a 3 horas, teniendo un nivel de dificultad intermedio. ¡Éxito en su implementación!

🔗 📄 🗨️ 🔄 🗑️

Ask anything

+ 🔍 Search 🔍 Reason

