Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Programación de Redes 1

Obligatorio 2

Renzo José Nro. Est. 240272

Nicolas Duarte Nro. Est. 268772

Repositorio: https://github.com/ArqSis-PDR-2024-2/obl-n6a-licenciatura-268772-165010-240272/tree/main

Grupo: N6A

Docentes: Andrés Soria y Guillermo Echichure

TABLA DE CONTENIDO

DESCRIPCIÓN DEL PROYECTO:

Presentamos la evolución del "Sistema Vapor", o "Steam System", una aplicación diseñada para la compra y publicación de videojuegos bajo una arquitectura cliente-servidor. En esta nueva versión, se ha migrado toda la estructura de hilos (Threads) hacia un modelo asincrónico utilizando async y await, mejorando así la eficiencia en la gestión de múltiples clientes simultáneos. Este cambio permite que el sistema maneje más usuarios sin comprometer la estabilidad ni el rendimiento del servidor, especialmente bajo cargas elevadas de conexiones.

Mejoras en el Manejo de Conexiones:

Para gestionar eficientemente las conexiones, ahora el sistema utiliza TcpListener y TcpClient, librerías de alto nivel que permiten una comunicación más robusta y confiable entre clientes y servidor. Cada cliente que se conecta al servidor dispone de un canal de comunicación dedicado que utiliza tareas (Tasks) asíncronas en lugar de hilos tradicionales, permitiendo que el servidor gestione las solicitudes de forma concurrente sin bloquear el flujo de trabajo.

Interfaz de Usuario y Modularidad:

Actualmente, la interfaz de usuario se basa en una línea de comandos, pero el sistema está diseñado con modularidad y flexibilidad en mente, lo que facilita futuras expansiones. Gracias a esta estructura, es posible incorporar una interfaz gráfica o métodos de interacción avanzados de manera sencilla y eficiente.

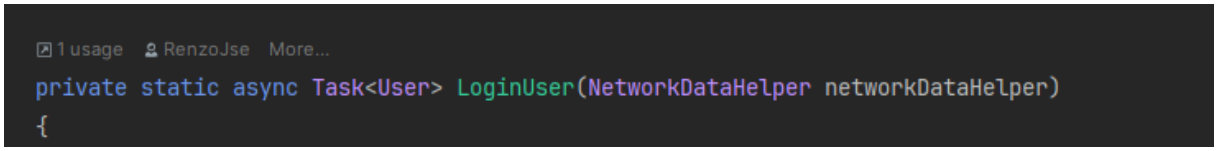
Funcionalidades Principales

Las funcionalidades clave de "Sistema Vapor" incluyen la publicación, compra, modificación, eliminación y búsqueda de videojuegos dentro del catálogo. En esta versión, el servidor está optimizado para manejar estas operaciones de manera concurrente y segura, beneficiándose de la implementación de async/await que permite mejorar el rendimiento y escalabilidad del sistema.

Mejoras de SteamSystem:

En respuesta a los requerimientos de la segunda entrega, el sistema ha sido migrado completamente a un modelo asincrónico basado en async y await. Las siguientes modificaciones fueron implementadas para cumplir con los objetivos del curso de Programación de Redes:

1. **Migración a Async/Await**: Toda la estructura orientada a hilos fue reemplazada por Tasks en base al modelo asincrónico de .NET, mejorando así la escalabilidad y la capacidad de respuesta del sistema.



2. **Uso de TcpListener y TcpClient**: La comunicación ahora utiliza las librerías TcpListener y TcpClient de alto nivel en lugar de sockets crudos, lo que proporciona un manejo más robusto y simplificado de las conexiones.

```
private static List<TcpClient> ConnectedClients = [];
```

```
while (_serverRunning)
    try
    {
        var programInstance = new Program();
        var client = await server.AcceptTcpClientAsync(); // El accept es blo
        lock (_lock)
        {
            ConnectedClients.Add(client);
        }
        Console.WriteLine("Client connected");
        var task = Task.Run(async () =>
            await HandleClient(client, programInstance, CancellationToken));
    }
    catch (Exception ex)
    {
        Console.WriteLine("Server has been shut down.");
    }
```

3. **Requerimientos Adicionales**:
    - Se implementó una opción para permitir cerrar el servidor, dando al administrador la posibilidad de esperar a que los clientes actuales se desconecten de forma natural o desconectarlos forzosamente, utilizando cancelación de tareas para un manejo adecuado de la desconexión.

```
private static readonly CancellationTokenSource CancellationTokenSource = new CancellationTokenSource();
private static readonly CancellationToken CancellationToken = CancellationTokenSource.Token;
```

```
if (command == "shutdown")
{
    Console.WriteLine("Do you want to wait for clients to disconnect? (yes/no)");
    var response:string? = Console.ReadLine();
    if (response?.ToLower() == "yes")
    {
        Console.WriteLine("Waiting for clients to disconnect...");

        while (ConnectedClients.Count > 0)
        {
            await Task.Delay(100); // Small delay to prevent busy-waiting
        }

        var disconnectTasks:List<Task> = ConnectedClients.Select(async client =>
        {
            client.Close();
        }).ToList();

        await Task.WhenAll(disconnectTasks);
        Console.WriteLine("All clients disconnected. Server is shutting down...");
        _serverRunning = false;
        server.Stop();
    }
```

En la siguiente imagen se puede observar como fue implementado este nuevo requerimiento utilizando beneficios como los CancellationToken y CancellationTokenSource. Para que funcione correctamente es necesario crear un await Task Delay debido a que los clientes pueden estar realizando consultas o utilizando el sistema mientras este se quiere apagar y de esta manera, se actualiza constantemente verificando si sigue estando conectado.

Beneficios de las Modificaciones:

Las modificaciones realizadas mejoran significativamente la eficiencia y capacidad del sistema para gestionar múltiples conexiones, brindando una experiencia de usuario más fluida y reduciendo la latencia en la respuesta del servidor. La adopción de async/await y las librerías de alto nivel también reduce la complejidad del código y facilita el mantenimiento, así como la integración de futuras mejoras.

Errores corregidos:

Se corrigió un error documentado en la entrega anterior, relacionado con la funcionalidad de modificar un juego específico. El problema ocurría al intentar cambiar la imagen de la carátula, lo cual generaba

una excepción en la clase NetworkDataHelper. Esta corrección asegura que el proceso de modificación de la imagen se realice sin interrupciones, mejorando la estabilidad y confiabilidad del sistema.

Además, se corrigió un error inesperado que surgió durante la defensa de la entrega. Este error, que no había sido previsto, ocurría al intentar obtener la información de un juego. La aplicación fallaba al intentar cargar la imagen del juego o cuando esta no estaba disponible. Esta corrección asegura que la obtención de la información del juego funcione correctamente tanto si la imagen existe como si no.