Programación de redes

Obligatorio 2

Gastón Landeira **- 238473**
Nicolás Rosa **- 225609**

Universidad ORT Uruguay

Profesor: Luis Barrague

Gastón Landeira - 238473
Nicolás Rosa - 225609

Índice

Notas:

-Todo el proyecto se realizó en Github, se puede encontrar en el siguiente link: https://github.com/nicoRosa18/Obligatorio-2-Programacion-de-redes

-No se agregaron diagramas, son iguales a la entrega pasada excepto por los métodos que implementan Tasks a los que se le agrega el nombre Async al final.

Migración a tcp

Se le añadió una capa más alta al manejo de sockets al cambiar los sockets en sí por Tcp. Cambios grandes no se tuvieron que hacer más que diferenciar los sockets en cliente y servidor y cambiar su tipo por TcpClient y TcpListener en toda parte de código.

Una parte que si noto cambio es la clase CommunicationSocket, ahora llamada CommunicationTcp, dentro del paquete communicator del proyecto Common. Se tuvo que agregar un NetworkStream que se obtiene de la conexión del TcpClient con el TcpListener, que permite el pasaje de datos. Los métodos de este objeto (Write y Read) sustituyeron los métodos que se obtenían del socket (Send y Receive).

Otro peculiar cambio surgió en ClientManager, que se vio forzado a agregar la ip y el puerto desde donde se conecta el TcpClient para poder crearlo, antes solo con la del servidor bastaba para conectar el socket. Estos se deben agregar desde el App.config del cliente.

Por último, la idea del fake socket se mantuvo y ahora es llamado fakeTcp.

Migración a async/await

Se migró del uso de Threads al modelo asincrónico de .NET. Este modelo supone el uso de Tasks los cuales cambian la firma de los métodos que implementan asincronía, por lo tanto se decidió diferenciar estos métodos y renombrarlos a NombreDelMetodo + Async (Menos los métodos Main).
Además, todos los métodos que implementan Task y devuelven un valor lo deben hacer por medio de un objeto Task, por lo tanto se cambió el código para que se reciba este en los métodos sincrónicos y se desencapsule su resultado. En estos lugares de código se espera a que termine la asincronía mediante la llamada al método + .Result.
Como resultado toda la clase CommunicatorTcp y el manejo de hilos en el servidor en ServerManager pasaron sus métodos a async. Esto permitió que la comunicación y conexión entre cliente y servidor sean asincrónicas pero, trajo consigo problemas con el manejo de excepciones y la finalización del servidor.

La naturalidad de las excepciones que son creadas dentro de métodos asincrónicos provocó que se cambiara la forma de atajarlas. Estas excepciones son contenidas en una excepción que actúa como un contenedor de excepciones llamado AggregateExceptions. Por lo tanto a los métodos asincrónicos que pueden fallar se les ataja esta excepción contenedora y posteriormente se las trata de forma independiente a cada una.

Al enviar la imagen de un archivo nos dimos cuenta que si el archivo no existe, el error no se propagaba al código principal. Esto es porque una excepción dentro de un método async await no se propaga al resto del sistema como antes (sincrónicamente). Lo que sucede es que el código sigue ejecutando y la excepción sucede después del try and catch del código principal. Por lo tanto se decidió hacer .Wait() a las llamadas del método que envía archivos (SendFileAsync) para poder chequear las excepciones que este levanta.

Al final del servidor (cuando se escribe exit por consola en el servidor) se realiza un WaitAll del task que maneja todas las conexiones con los clientes para terminar de manera prolija la ejecución del proyecto.

Por último, se menciona que surgieron 21 warnings nuevos en esta entrega respecto a la pasada. Estos son la misma advertencia todas las veces que avisa que las llamadas a los métodos asincrónicos de comunicación no son esperados (awaited). Esto no es necesario porque cada vez que se manda un mensaje siempre se espera una respuesta (en el cliente), por lo tanto se espera a que terminen los métodos asincrónicos al esperar por la respuesta del mensaje con .Result mencionado anteriormente.