Universidad ORT Uruguay

Facultad de Ingeniería

Programación de redes
Obligatorio 2

Jimena Sanguinett(228322)
Joselen Cecilia(233552)

Entregado como requisito de la materia Programación de redes

15 de mayo de 2021

Declaraciones de autoría

Nosotras, Joselen Cecilia y Jimena Sanguinetti, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos Programación de redes ;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Índice general

1. Introduccion

Se construye un sistema formado por dos aplicaciones de consola con un menú cada una, cuenta con un servidor en el que se deben guardar datos de posts de usuarios los cuales contienen temas y archivos. Además cuenta con un cliente para dicho servidor que se encargará de permitir interactuar a los clientes con el sistema.

El sistema cumple con las siguientes funcionalidades:

Del lado servidor:

-Aceptar pedidos de conexión de un cliente
-Mostrar los clientes conectados
-Listado de temas
-Listado de posts por tema
-Ver un post especifico
-Ver archivo asociado a un post especifico
-Ver tema con más posts
-Ver listado de archivos

Del lado del cliente:

-Conectarse y desconectarse al servidor
-Alta de tema
-Baja y modificación de tema
-Alta de post
-Baja y modificación de post
-Asociar y desasociar post a un tema
-Subir archivos relacionados con el post

2. Alcance

El alcance de esta segunda entrega, en cuanto a la estructura general, no recibió cambios respecto a la primera entrega.

3. Descripción de la arquitectura

Task:

Tanto en la aplicación del cliente como del servidor fue necesario el uso de Task, esto se hizo para poder mantener varios subprocesos a la vez de forma asincrónica. Para esto fue necesario modificar la firma de los métodos a .ªsync Task.º .ªsync Task¡T¿z a la hora de invocarlos se utilizó el .ªwait"; además debimos cambiar tanto los métodos de envío y recibo de data como los de conexión a tareas asincrónicas, por ejemplo, readAsync y acceptTcpClientAsync.

async/await:

Mediante el await se puede esperar el resultado de las async Tasks ya que con este se pausa la ejecución hasta que termine la tarea y se obtenga un resultado.

Stream:

Para poder enviar y recibir data, ya sean packet o arcivos fue necesario el uso de la clase NetworkStream de modo que se pueda acceder a los métodos y propiedades de la clase Stream (clase padre) que se utilizan para facilitar las comunicaciones de red. Una vez que se obtiene una instancia de esta clase, se llama al método Write para enviar data al servidor, mientras que el servidor llama al método Read para poder recibir la data que llega. Ambos métodos se bloquean mientras se esta realizando la operación; la operación de lectura se desbloquea cuando los datos han llegado y están disponibles para ser leídos.

TcpClient y TcpListner:

Para establecer la conexión y luego enviar y recibir datos entre varios clientes y el servidor utilizamos la clase TcpClient la cual nos proporciona los métodos necesarios para esto.
Luego para aceptar las conexiones enviadas por el cliente de forma asincrónica utilizamos el método AcceptTcpClientAsync de la clase TcpListener.

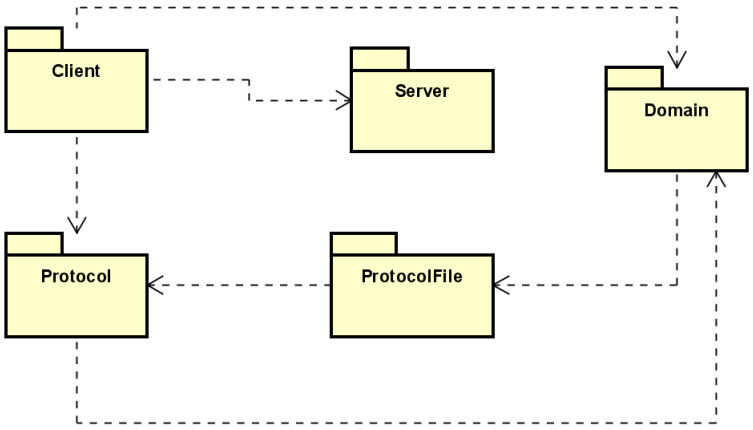
Semaphore Slim:

Una mejora con respecto a la entrega anterior fue el uso de semáforos, como documentamos anteriormente, la forma de validar que un Post y un Theme no sean

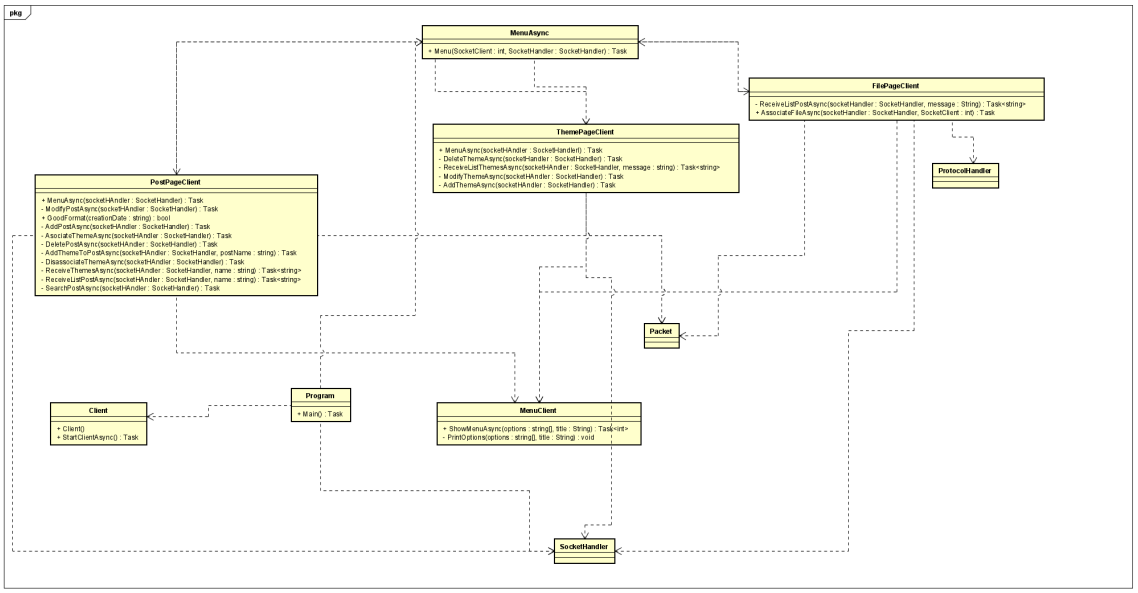
modificados mientras otro lo está modificando se realizó mediante un bool que indicaba si estaba en uso o no. Para esta segunda entrega agregamos dos clases nuevas: SemaphoreSlimPost y SemaphoreSlimTheme en las cuales tenemos un semáforo y un Post o Theme (según corresponda). Por otro lado, en la clase MemoryRepository se agregan dos listas nuevas: SemaphoreSlimPosts y SemaphoreSlimThemes que van a contener los semáforos correspondientes a cada Post y Theme en uso. A la hora de modificar o eliminar uno de estos, primero se chequea su disponibilidad, en caso de que el semáforo no exista (dado que se crea la primera vez que se va a modificar) o se encuentre en 1, se puede realizar cambios en el pero si este se encuentra en 0 quiere decir que alguien lo esta utilizando por lo que debe esperar a que sea liberado, lo cual sucede una vez que se deja de usar. Al poder utilizarlo uno a la vez, ambos semáforos tienen disponibilidad 1, es por eso que el chequeo es entre 1 y 0.

4. Diseño detallado de cada uno de sus componentes

El diseño de nuestra aplicación se basa en varias capas las cuales no recibieron grandes cambios respecto a la primera entrega, de todos modos podemos ver los mismos a continuación, con los respectivos nuevos diagramas:

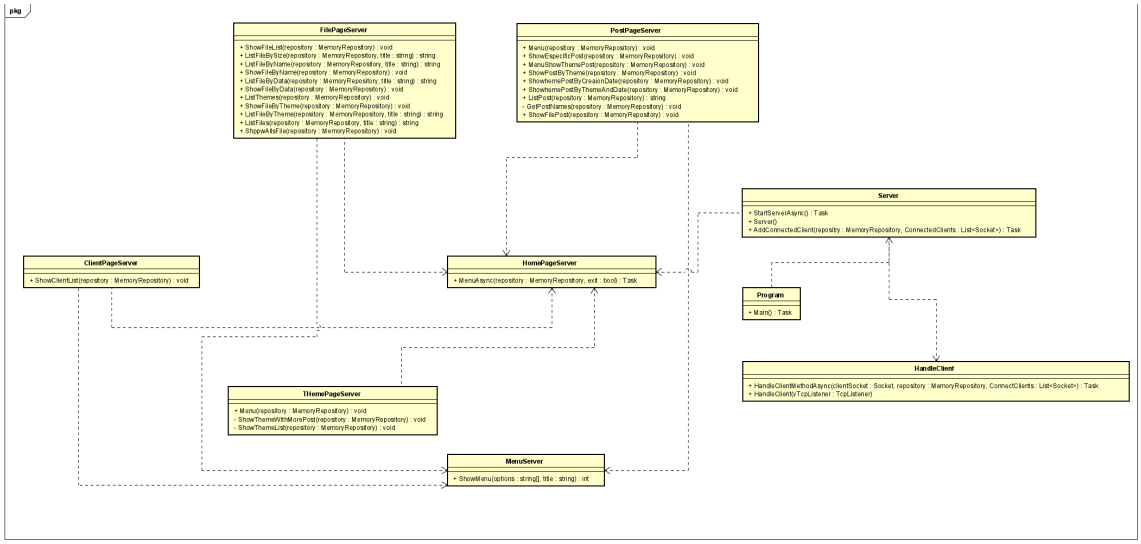


1. Client:



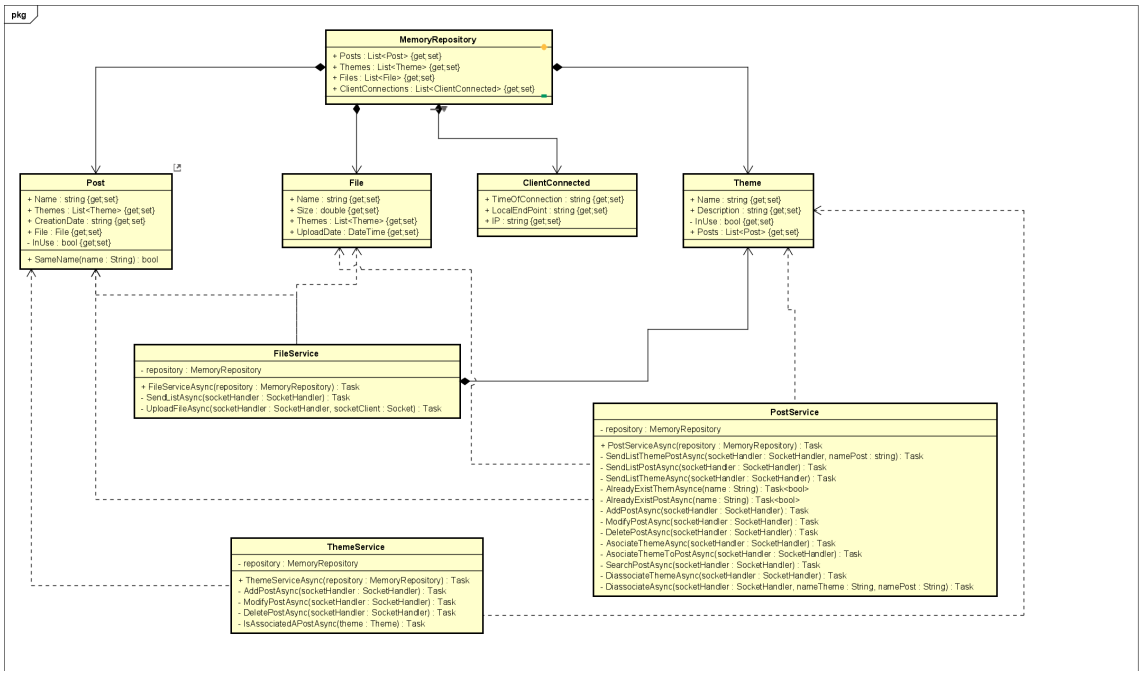
Dentro de esta capa, uno de los mayores cambios recibidos fue que en la clase ConnectionClient pasó a ser simplemente Client dado que se utiliza TcpClient y ahora solo inicia la conexión. Así mismo, la firma de los métodos propios fueron modificadas, dado que ahora son .ªsync Task.º .ªsync Task¡T¿z la invocación a los mismos es mediante el uso del await; los métodos propios de la conexión, el envío y la recepción de datos también se modificaron dado que ahora son async. Por último, ahora el socketHanlder recibe el tcpClient.GetStream() el cual le permite acceder a todas las operaciones necesarias.

2. Server:



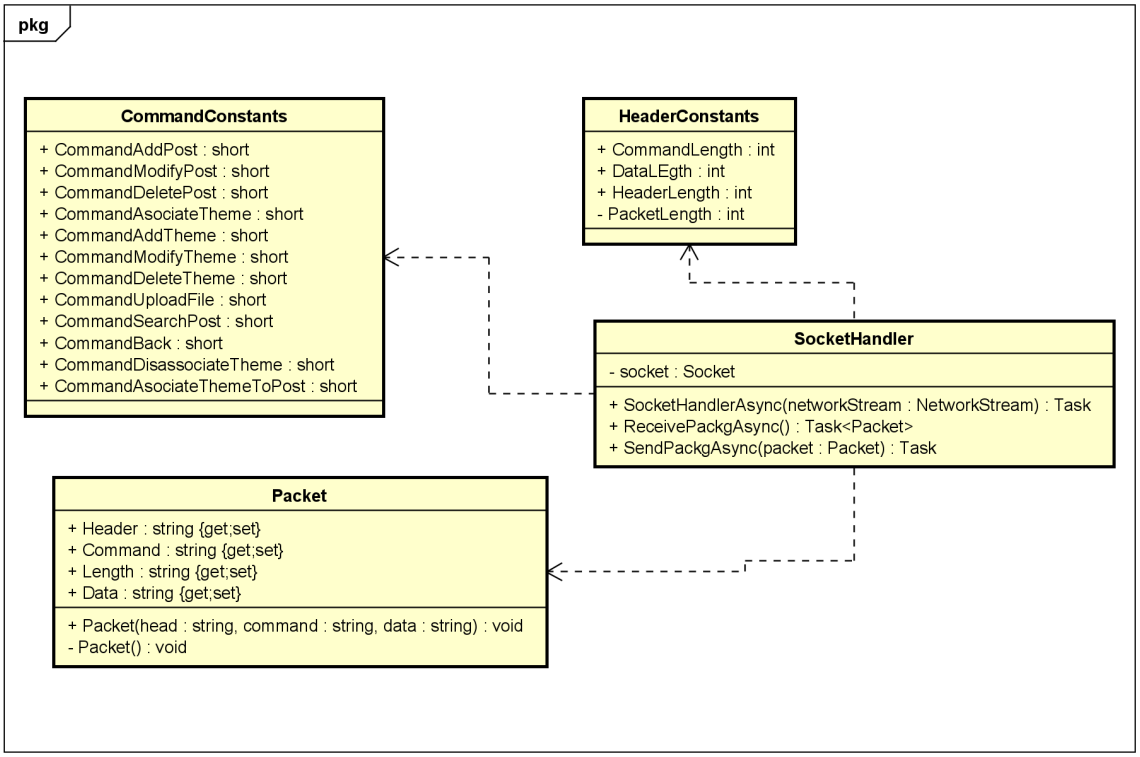
Del lado del servidor se recibieron menos cambios, ahora contamos con la clase Server la cual es la encargada de iniciar el server y agregar las conexiones de los diferentes clientes mediante el tcpListener. Además, al ser removidos los hilos ahora se utiliza el Task.Run() para mostrar el menú.

3. Domain:



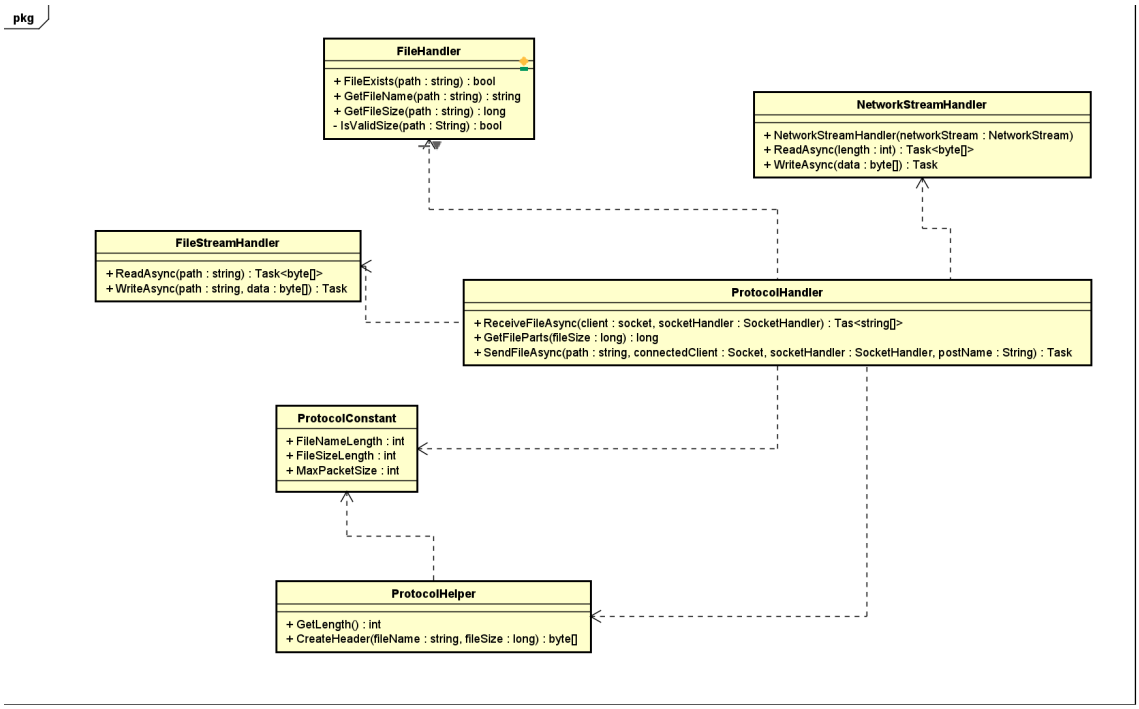
El único cambio realizado fue que en los services los métodos son asincrónicos.

4. Protocol:



Del funcionamiento del protocolo en sí no hubo modificaciones, lo que cambió fue nuevamente que el socketHandler recibe el NetworkStream el cual luego me permite utilizar el ReadAsync y el WriteAsync.

5. ProtocolFile:



En ProtocolFile ahora los archivos se envían y se recibe por partes dado que así se requiere en la letra.

Nota: en caso de querer ver los UML con mejor calidad, se encuentran adjuntados en la carpeta del obligatorio dentro de UMLs

5. Documentación de mecanismos de comunicación de los componentes de su solución

5.1. Protocolo

Para que el Servidor y el Cliente puedan comunicarse de forma que el servidor reciba las requests y envíe las respuestas para que luego el cliente las interprete, fue necesario definir dos protocolos, uno para el envío de strings (Protocol) y otro para el envío de archivos (ProtocolFile).

Protocol

Para el envío y recibo de la información, se crearon dos métodos writePakgAsync y ReceivePakgAsync los cuales se encargan de enviar y recibir los paquetes con la información.

Estos paquetes están formados por un header, un command, un length y un data; el header nos indica si el mensaje se manda por una response o una request (RES/REQ), este tiene un largo fijo de3 bytes; el command es el número del requerimiento, que me indica cual funcionalidad debo llamar y tiene un largo fijo de 2 bytes, si el largo es de una cifra se rellena con ceros a la izquierda (ejemplo: 02).
El length me indica el largo de la data y se le suman cinco bytes(header+command) y tiene un largo fijo de 4, al igual que el command si la cifra del length es menor a 4 se rellena con ceros a la izquierda (ejemplo: 0012, indica que la data es de largo 12)
Por último, la data es un string que contiene toda la información que se esta enviando concatenada mediante #. Para completar el paquete.

Antes de mandar la información es necesario crear el Packet mientras que una vez que se recibe se debe hacer el proceso inverso para que se pueda interpretar la data del mismo. Este proceso inverso implica separar, según los #, la data del Packet, lo que nos permite acceder a la información en sí.
En el ReceivePakAsyn se leen los datos, con la función ReadAsync de networkStream, header, commmand y length almacenándolos en un array de largo 9 (3 para header, 2 para command y 4 para el length), que luego se convierte en string para

asignarlos al packet.command, packet.header y packet.length respectivamente.
Luego con ese length mas nueve (estos 9 son: 3 del header,2 del command y 4 del length) leemos la data, el mensaje que se envió tambien con la funcion ReadAsync de networkStream, almacenándola primero en un array de largo length y luego convirtiéndola en string para asignársela al packet.data.
En el SendPakgAsync se recibe un packet, se junta toda la información del packet en un string y se manda usando el WriteAsync de networkStream.

Nombre del campo	Header	Command	Length	Data
Valores	RES/REQ	1-12	0-9999	Data#data#...
Largo	3	2	4	Length

ProtocolFile

Para enviar y recibir archivos, se debe definir un FileNameLength, un FileSize y un MaxPacketSize; a partir de estos datos se define el paquete en el que se va a mandar. Este paquete se va a dividir en determinada cantidad de partes, dependiendo del MaxPacketSize ya que este es el tamaño máximo que puede tener.

Cuando el cliente envía el archivo utiliza el método WriteAsync de networkStream, invocado dentro del método SendFileAsync, escribe al mismo en partes.
Mientras que el servidor recibe el archivo llamando al método ReadAsync de networkStream invocado dentro del método ReceiveFileAsync, este también lee el archivo en partes
Dentro del envío y la recepción del archivo, también se incluye el envío de los datos del mismo (nombre, tamaño y Post al cual se va a asociar), como un paquete.

6. Manual de usuario

Luego de abierta la solución, se debe configurar en el App.config la IP correspondiente a la computadora del usuario.

Después de esta acción se debe ejecutar la aplicación de consola Server y luego la aplicación de consola Client.

Una vez ejecutadas las dos aplicaciones, el usuario verá dos menús, en los cuales deberá seleccionar la opción que desee.

Para seleccionar la opción el usuario debe desplazarse por el menú con las teclas down y up del teclado y luego al llegar a la opción que desee deberá presionar enter. Según la opción que seleccione se le mostrará la página que corresponda.

En los casos que se muestren las opciones con índices, se debe ingresar el número indicado en cada opción para poder acceder a la funcionalidad.

7. Link al repositorio de GitHub

https://github.com/JoselenC/Obligatorio1-Programacion-De-Redes.git