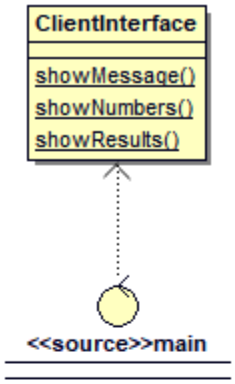
Ejercicio 1:

Diagrama general de la solución

Diagrama del lado del Cliente.



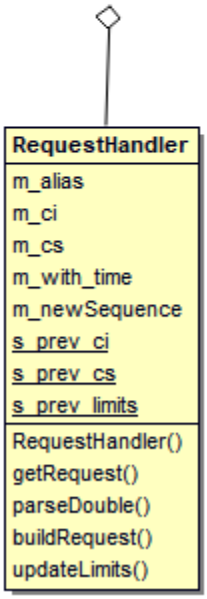
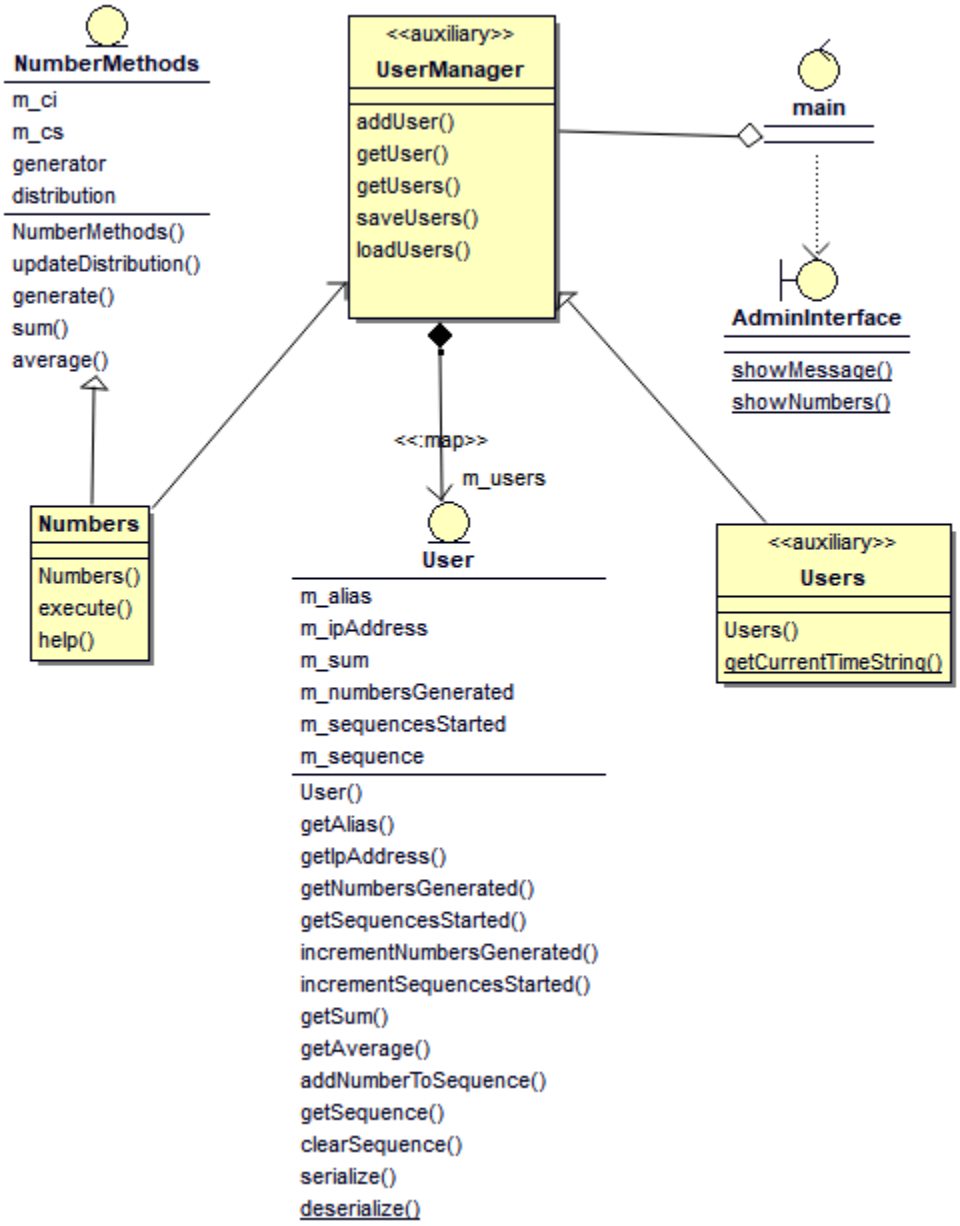
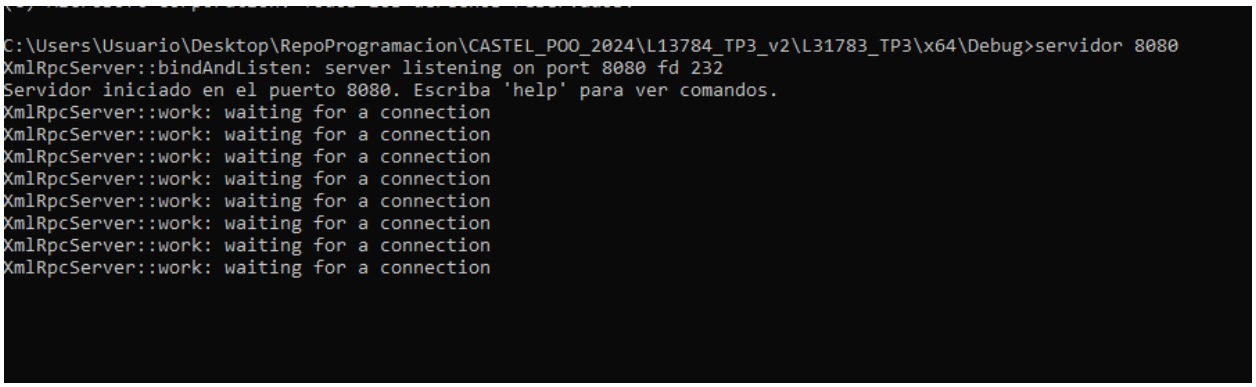


Diagrama de clases del lado del Servidor.



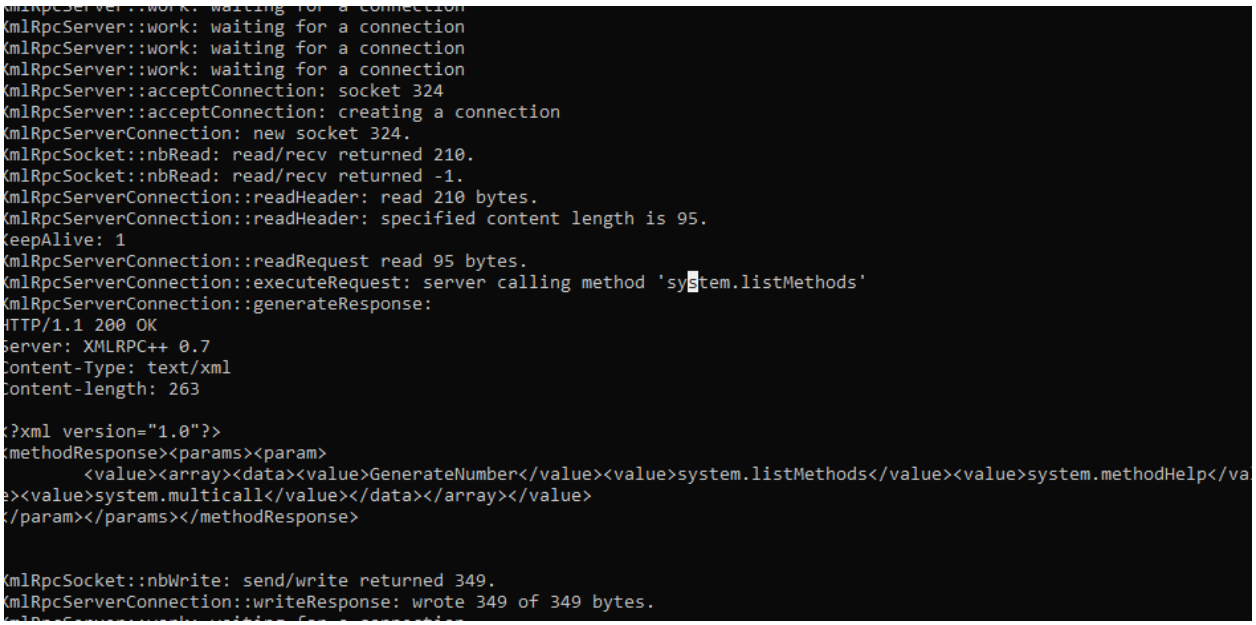
Modo de uso

Se debe ejecutar el programa **Servidor.exe** siendo el argumento de linea de comandos el puerto que quiera utilizarse para el server. Ejemplo: "*servidor 8080"*



Una vez iniciado, el servidor mostrará el mensaje "*waiting for a connection",* en ese momento podemos iniciar el programa **Cliente.exe** el cual debe ejecutarse con dos argumentos, el primero debe ser la ip local que es "**127.0.0.1**" y el segundo debe ser el puerto utilizado para iniciar el servidor. Ejemplo: "*cliente 127.0.0.1 8080".*

El servidor mostrará que la conexión se ha establecido devolviendo como prueba de vida los métodos disponibles, así como en el ejemplo que se envió (*hola_client.cpp y hola_server.cpp)*.



Del lado del cliente luego de ejecutarse, se mostrará lo siguiente:

```
<value><array><data><value>GenerateNumber</value><value>system.listMethods</value><value>
system.methodHelp</value><value>system.multicall</value></data></array></value>
Introduzca peticion (alias param1 param2 param3 param4)
```

Debajo se detallan las distintas combinaciones posibles de petición:

- ***alias NS ci cs T*** : el servidor inicia una secuencia nueva, estableciendo los límites ci y cs, además devolviendo fecha y hora de cuando fue generado.
- ***alias NS ci cs***: idem anterior pero sin fecha y hora.
- ***alias NS***: se inicia una nueva secuencia utilizando los límites de la petición anterior, si no hay ninguno establecido, se muestra al cliente un aviso de que deben ingresarse al menos para la primera petición.
- ***alias:*** si existe una secuencia iniciada, se continúa con ella y se utilizan los límites de la petición anterior.
- ***alias ci cs:*** idem anterior pero se establecen límites.
- ***alias T***: idem anterior pero no se establecen límites nuevos y se solicita fecha y hora.
- ***alias ci cs T***
- ***alias NS T***
- ***help***: se muestra al usuario la explicación de cómo y dónde usar NS y T.
- ***exit***

Ejemplo: Cliente envía tres veces la petición “usuario1 0 1000 T”.

```
Metodos:
 <value><array><data><value>GenerateNumber</value><value>system.listMethods</value><value>system.methodHelp</value><val
e>system.multicall</value></data></array></value>
Introduzca peticion, maximo 4 parametros. (alias param1 param2 param3 param4)
usuario1 0 1000 T
Fecha y hora: 2024-09-30 08:55:44
Numero generado: 924.47
Suma acumulada: 924.47
Promedio: 924.47
Numeros en la secuencia: 924.47

Introduzca peticion, maximo 4 parametros. (alias param1 param2 param3 param4)
usuario1 0 1000 T
Fecha y hora: 2024-09-30 08:55:47
Numero generado: 659.19
Suma acumulada: 1583.66
Promedio: 791.83
Numeros en la secuencia: 659.19 924.47

Introduzca peticion, maximo 4 parametros. (alias param1 param2 param3 param4)
usuario1 0 1000 T
Fecha y hora: 2024-09-30 08:55:48
Numero generado: 93.99
Suma acumulada: 1677.65
Promedio: 559.217
Numeros en la secuencia: 93.99 659.19 924.47

Introduzca peticion, maximo 4 parametros. (alias param1 param2 param3 param4)
```

Ejemplo: el usuario1 realiza una petición, luego el usuario2, y por ultimo nuevamente el usuario1.

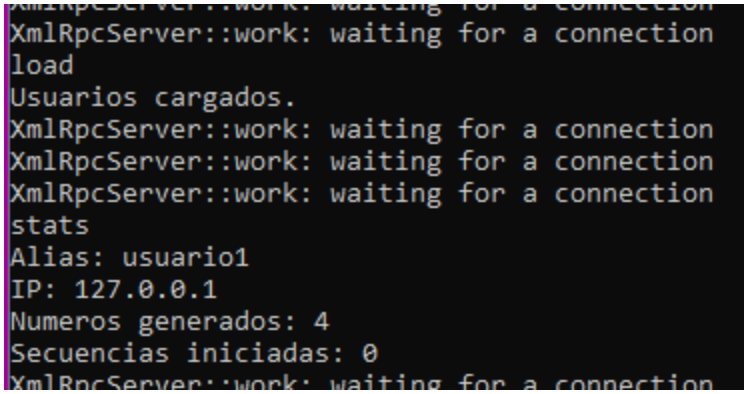
```
Introduzca peticion, maximo 4 parametros. (alias param1 param2 param3 param4)
usuario1 NS 0 1000 T
Fecha y hora: 2024-09-30 08:57:54
Numero generado: 438.05
Suma acumulada: 438.05
Promedio: 438.05
Numeros en la secuencia: 438.05

Introduzca peticion, maximo 4 parametros. (alias param1 param2 param3 param4)
usuario2
Numero generado: 467.63
Suma acumulada: 467.63
Promedio: 467.63
Numeros en la secuencia: 467.63

Introduzca peticion, maximo 4 parametros. (alias param1 param2 param3 param4)
usuario1 0 1000 T
Fecha y hora: 2024-09-30 08:58:19
Numero generado: 558.93
Suma acumulada: 996.98
Promedio: 498.49
Numeros en la secuencia: 438.05 558.93

Introduzca peticion, maximo 4 parametros. (alias param1 param2 param3 param4)
```

El servidor siempre pedirá un alias para verificar la petición y poder relacionar los números generados y secuencias generadas a cada uno de ellos.

Ahora del lado del servidor, el administrador también puede realizar ciertas acciones, la lista de comandos es la siguiente:

```
XmlRpcServer::work: waiting for a connection
help
Comandos disponibles:
exit: salir del servidor
save: guardar los usuarios en un archivo
load: cargar los usuarios de un archivo
stats: mostrar estadisticas de los usuarios
clear: borrar contenido de archivo users.bin
XmlRpcServer::work: waiting for a connection
XmlRpcServer::work: waiting for a connection
```

Si se piden estadísticas y el servidor todavía no ha recibido peticiones, no se mostrará nada. Pueden visualizarse las estadísticas de los usuarios de la última conexión del servidor, mediante el comando “load”, donde se mostrará por ejemplo:



El servidor serializa los usuarios como objetos, incluyendo en ellos información como la IP desde donde se conectó, la cantidad de números generados, y la cantidad de secuencias iniciadas.

Ejemplo: Si el servidor inicia, y el cliente con alias: usuario1 comienza a realizar peticiones, el administrador tiene la opción de usar el comando “load” y si en el archivo “users.bin” existía el usuario, se priorizará la información del archivo. El administrador puede elegir sobrescribir “users.bin” con el comando “save”.

El comando “save” sobrescribe la información del archivo “users.bin” con la de la ejecución actual. Luego en una próxima ejecución, puede utilizarse el comando “load” y si el servidor recibe peticiones que coincidan con alguno de los usuarios que se han cargado, se sumarán las estadísticas correspondientes, **de todas formas, para la comodidad del administrador, siempre que se inicia el servidor se cargan los usuarios existentes en el archivo y siempre que se cierra se guardan.**

Comentarios

Hay varios detalles de este proyecto que son importantes de comentar y aclarar, el primero es que aunque no se pidió un manejo de usuarios, se concluyó que era enriquecedor el hecho de implementarlo, además gracias a ello, fue posible incluir serialización de objetos, aunque simple, es interesante como se puede guardar el objeto “usuario” y no un simple alias, o un simple vector que tiene asignado cada valor.

Otra cosa que es importante aclarar es la versatilidad de las peticiones, se perdió mucho tiempo y recursos en hacer a prueba de errores las peticiones del cliente, siendo que quizás podría haberse elegido otro formato/protocolo, aun así se concluyó que era interesante jugar con las posibilidades del cliente y darle mucha libertad al momento de realizar la petición, esto está ideado basándose en cómo funciona un sistema C/S real, donde el cliente puede realizar peticiones desde distintos usuarios (distintas aplicaciones) y donde cada aplicación tiene sus parametros especificos.

Se utilizó por primera vez la función **_kbhit()** la cual permitió, dado el hecho de que no se ha utilizado multithreading, de atender a la terminal cuando se haga click en ella para escribir algún comando, si no es así, el servidor se mantiene en escucha cada 400 ms.

```
if (_kbhit()) {
    std::getline(std::cin, command);
```

Breve conclusión

En general el proyecto satisface las necesidades pedidas, aunque seguramente quedan errores por cubrir, y manejo de excepciones por suplir, pero una conclusión interesante que será tenida en cuenta para el proyecto integrador, es el hecho de que el servidor para permanecer en estado de escucha lo más cercano a *constante* debe utilizarse alguna forma de multithreading para atender a los comandos del administrador y al mismo tiempo a las peticiones del cliente. Aunque con kbhit() se cubre bastante bien este problema, lleva a un nivel más allá el proyecto la utilización de al menos dos hilos, siendo que la implementación de esta característica no implica grandes esfuerzos.

Un detalle que no se tuvo en cuenta fue el hecho de hacer todavía más user friendly la aplicación de cliente, por ejemplo, se podría haber añadido la característica de no ser *case-sensitive* a los comandos.

Por último no está de más decir que este proyecto fue muy interesante, de los más largos que he realizado y sin dudas he aprendido bastante sobre el lenguaje y el cómo funciona una aplicación C/S.

Recursos adicionales

Se recomienda encarecidamente utilizar la librería que se proporciona en el .zip enviado. La, librería es la misma que la proporcionada por la cátedra, pero ha sido compilada en un .lib y además han sido agregadas ciertas macros a los archivos, las cuales difieren del instructivo proporcionado junto con ellas, debería poderse compilar y ejecutar si se abre en cualquier entorno.