**Universidad Nacional de la Patagonia San Juan Bosco**

**Licenciatura en Sistemas O.P.C.G.P.I.**

****

**Sistemas Distribuidos**

**TP N.º 1**

**Docentes: Mg. Ing. Ricardo Antonio López**

**Lic. Cristian Javier Parise**

**Alumnos: Aguila Maximiliano**

**Krmpotic Lucas**

**Año 2018**

**Trabajo de Laboratorio 1**

**Cliente / Servidor. Sockets. RPC. Threads. Concurrencia. RFS.**

1. Tome el código provisto en la carpeta p11.

a) Analice los fuentes client.c y server.c y modifíquelos para que la consulta del cliente y la

respuesta del servidor sea más interactiva (hacerlo iterativo, cambiando el texto y devolverlo alcliente, etc).

\_ Se modificaron los archivos fuente (server.c y client.c) de manera tal que el cliente solicite al servidor realizar una suma de una serie de numeros enteros (separados por espacios).

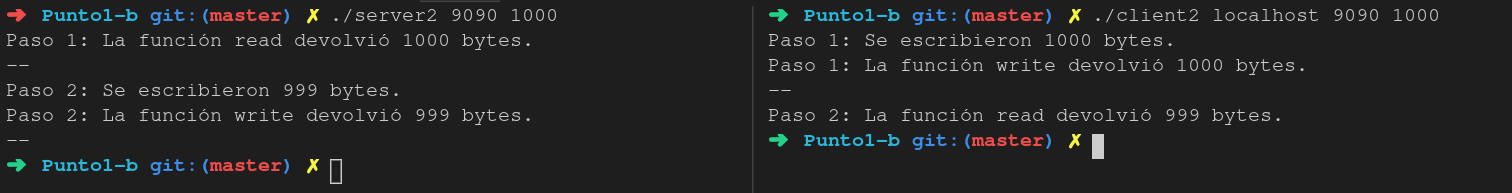
Luego el servidor toma esa cadena y la tokeniza (utilizando la funcion de *strtok*) y encuentra los distintos numeros enteros de la serie (a cada token le aplica la funcion *atoi*) aplica la funcion solicitada y luego le devuelve al cliente el resultado de la suma de la serie.

\_Los archivos modificados estan incluidos en el comprimido en la carpeta Punto1-a

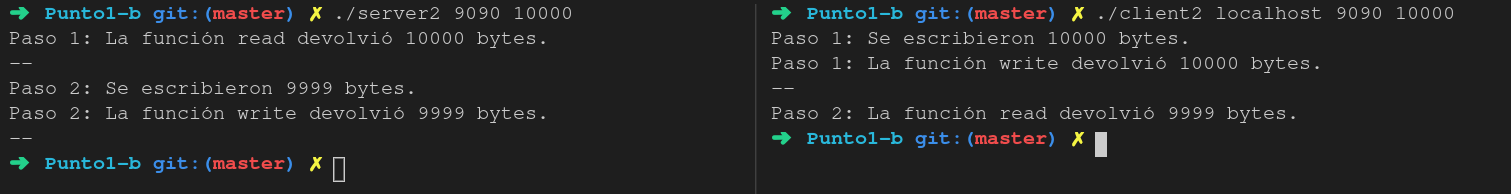
b) Analice los fuentes client2.c y server2.c para ver su funcionamiento. Modifique el tamaño de los buffers para que sean de longitud fija: 10 3 , 10 4 , 10 5 y 10 6 bytes. Explique las diferencias obtenidas al ejecutar en cada caso.

\_ Se realizaron diferentes capturas para verificar su funcionamiento:

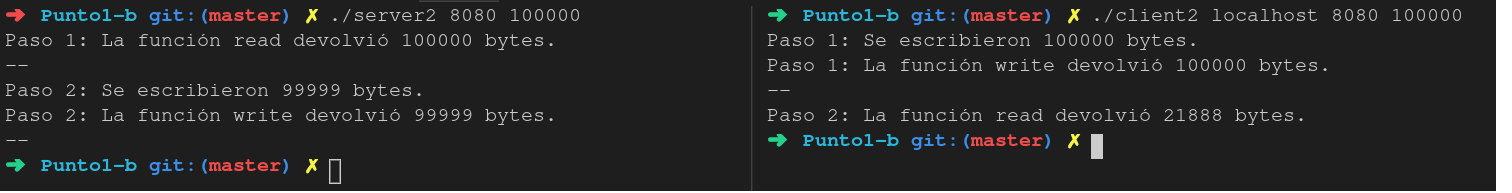
Longitud 103



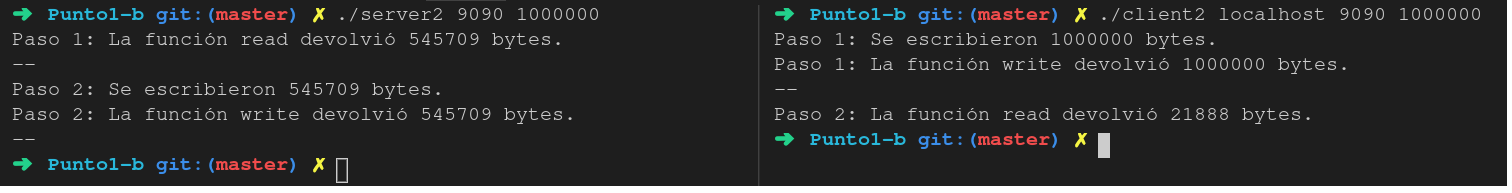
Longitud 104



Longitud 105



Longitud 106



\_ El formato de la llamada a sistema de escritura es:

*ssize\_t write(int fd, const void \*buf, size\_t count);*

En caso de éxito, se devuelve el número de bytes escritos (el cero indica que no se ha escrito nada). En caso de error, se devuelve -1 y errnose establece de forma adecuada.

Sin embargo, no dice cuál es el límite del parámetro de conteo. Todavía no dice el comportamiento cuando el recuento es mayor que SSIZE\_MAX.

Teniendo en cuenta que escribir es una llamada al sistema que se puede usar para dispositivos / archivos genéricos / lo que sea, si el dispositivo admite operaciones de escritura mayores que SSIZE\_MAX, el tipo de devolución no puede manejar la cantidad real de bytes escritos.

Estos resultados son consecuencia de que la cantidad de bytes leidas y devueltas

(de tipo ssize\_t) superan el máximo preestablecido indicando en la constante

SSIZE\_MAX.

2. Tome el código provisto en la carpeta p12.

a) Analice el código provisto en la carpeta p12 para luego responder las siguientes consignas:

1) Definir brevemente qué es un servidor con estados y qué es un servidor sin estados.

\_ Un servidor con estados es aquel que va manteniendo información (estados) sobre las diferentes operaciones que ha realizado (sesiones). El resguardo de estos

estados permite que el cliente tenga que transmitir menor cantidad de información puesto que no debe reiterar aquellos datos no variantes (numero de file sistem por ejemplo). Los estados guardados en

un servidor con estados deben ser actualizada conforme estos varían.

Por otro lado un servidor sin estados es aquel que no mantiene información sobre las actividades que fueron realizando. De este modo, por cada operación, el cliente debe volver a brindar los datos que son necesarios, provocando

mayor tráfico de datos y sin tener un "historial" de las sesiones u operaciones realizadas anteriormente.

2) Explicar si el servidor implementado en p12 es de la clase de servidores con o sin estado.

\_ El servidor corresponde al tipo de servidores con estados, ya que mantiene

información sobre los archivos abiertos. Cada vez que se desea leer o escribir un

archivo primero se le solicita al servidor que lo abra. El servidor abre dicho archivo y

devuelve al cliente el file descriptor del mismo para las posteriores operaciones.

b) Tomando el código analizado documente los pasos (y si es pertinente incluya código o

pseudocódigo) para implementar un servidor opuesto al ya implementado (Si es sin estado

realice uno con estado o viceversa) con la misma funcionalidad general, probablemente se vea alterada la interfaz del servicio, pero que semánticamente brinde el mismo servicio.

\_ Para que el servidor pase de ser un servidor con estado (statefull) a uno sin estado (stateless) se deben primero incorporar las funciones atomicas de "opening" y "closing" de archivos en las funciones de lectura y escritura.

Asi la funcion "rmread" y "rmwrite" deberan recibir todos los parametros necesarios para la apertura y cierre de archivos.

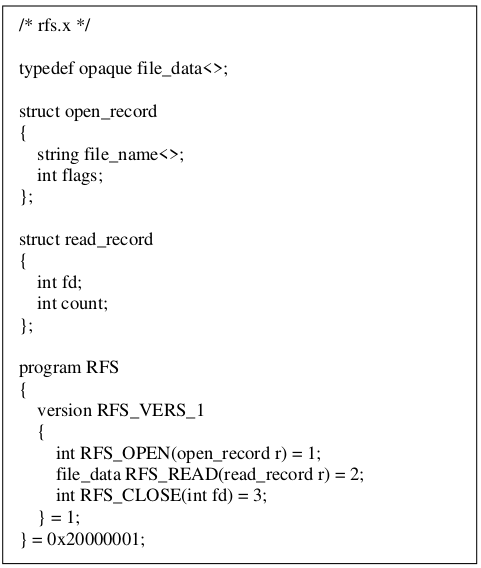
A nivel Middleware, las cuatro funciones que había previamente (para open, close, send y receive)pasaron a ser solo dos, eliminando los correspondientes a apertura y cerrado en ambos stubs,

teniendo asi que modificar las estructuras correspondientes para el uso de los punteros a función.

Luego de estas modificaciones, al eliminar las operaciones de apertura y cierre de archivos, y que solo existan operaciones de lectura y escritura, tambien podrian eliminarse el pasaje de las flags, ya que

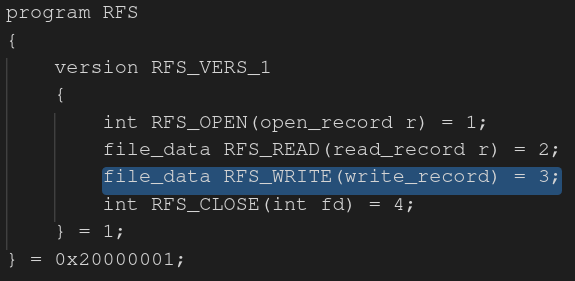
en el momento de abrir un archivo, la funcion ya sabria que proposito tiene, pudiendo adecuar mejor los parametros enviados.

3. Dada la siguiente especificación RPC:

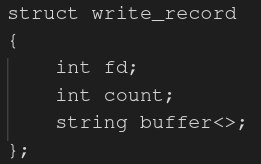


a) Agregue la operación RFS\_WRITE al .x de la especificación e implemente la solución completa (tener en cuenta que gran parte de la codificación ya se encuentra en apunte S21-Rpc-Rfs-v2.pdf).

\_ Se agrego la operación RFS\_WRITE al .x



Tambien se definio la estructura “*write\_record*”



b) Implemente el equivalente en Java con sockets (tener en cuenta apunte S31-Java sockets y threads-v4.pdf). Se pide en este caso que el cliente cuenta con interfaz gráfica.

c) Comparar y comentar la complejidad de ambas implementaciones.

4. Teniendo en cuenta el servidor del ejercicio anterior, 3.b:

a) Pruebe cancelar un cliente cuando se está en el medio del funcionamiento y vuelva a arrancar el cliente.

Observe y documente qué ocurre.

b) Pruebe cancelar el servidor cuando se está en el medio del funcionamiento y vuelva a arrancar el cliente.

Observe y documente qué ocurre.

c) Determine si es un servidor con estados o sin estados.

5. Modifique la solución 3.b), y utilice los threads de Java para que se puedan responder

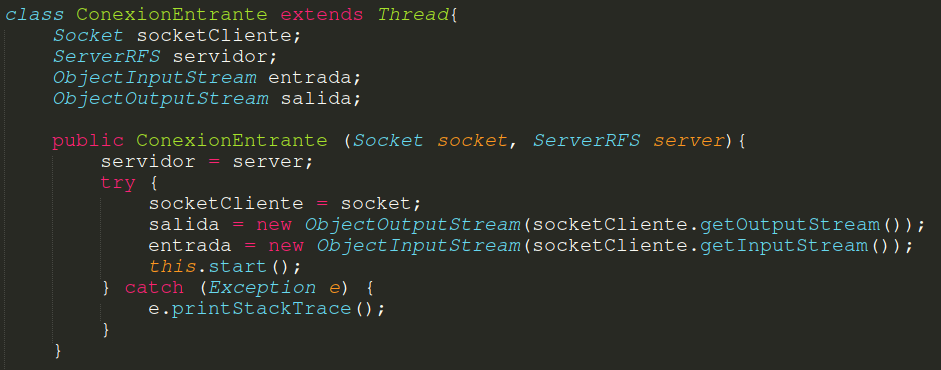
requerimientos de clientes de manera concurrente.

\_ Las modificaciones aplicadas para que el servidor pueda atender varios clientes concurrentemente fueron agregar una clase “ConexionEntrante” la cual hereda de “Threads” y que desde el Stub del servidor en vez de atender la conexión de un cliente, le crea un Hilo.

Cada hilo en su metodo “run” realiza las llamadas remotas.

De esta forma, cada cliente que se conecte tendrá su propia instancia del stub para realizar operaciones en su instancia de servidor.

Clase: “ConexionEntrante”



Clase: “ServerStub”



6. Concurrencia.

a) Disparar dos clientes a la vez y verificar (documentar) si las solicitudes de ambos clientes son

atendidas por el mismo o por distintos hilos en el servidor

b) Determine si la implementación del ejercicio 3.- tiene un thread por requerimiento, por conexión

o por recurso (en términos del cap. 6 de Coulouris).

c) Transferir un archivo de gran tamaño (GBs) completo en la versión con RPC y en la versión con Java Sockets (utilizando las primitivas de lectura o escritura con un tamaño de buffer igual en ambos casos) y cronometrar (que el mismo cliente muestre cuanto demoró) en ambos casos, sacando las conclusiones del caso respecto a la performance de cada solución.

7. Proponga una modificación del ejercicio 3.- de manera tal que se tenga un conjunto (pool) de threads creados con anterioridad a la llegada de los requerimientos y donde se administren los threads de acuerdo a las llegadas de requerimientos ¿Sería útil cambiar la cantidad de threads administrados de esta manera? Justifique.

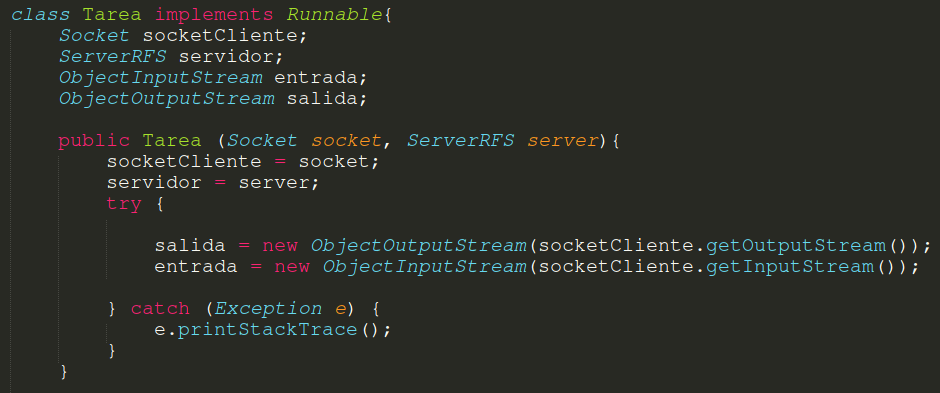
\_ Al utilizar un “pool” de tareas lo que se logra no es solo el control de una indiscriminada creacion y cierre de hilos (Threads) sino tambien asignarle una tarea previamente creada a cada cliente entrante.

Como desventaja tenemos que si un pool esta definido para 5 tareas, si existe un 6to cliente, este quedaria a la espera que alguna de las tareas del pool se desocupe.

\_ Las modificaciones implementadas son una nueva clase “Tarea” que debe implementar la interface Runnable.

Luego desde el “ServerStub” se crea el “pool” y por cada cliente entrante se le asigna una tarea.

Clase: “Tarea”



Clase: “ServerStub”

