Memoria técnica del proyecto

Programación de Sistemas Concurrentes y Distribuidos

2º curso, Grado de Ingeniería en Informática, 2018-2019

Número de equipo: *1*

Integrantes:

- *Germán Garcés Latre, 757024*
- *Sergio García Esteban, 755844*
- *Diego Caballé Casanova, 738712*
- *Patricia Briones Yus, 735576*

Índice de contenidos

1. Introducción

1.1 Descripción breve del trabajo

El proyecto consiste en el desarrollo de un servicio, un sistema de coordinación Linda. Linda es un lenguaje/sistema de coordinación basado en un espacio "lógico" de memoria compartida: el espacio de tuplas.

Su funcionamiento se basa en un proceso servidor, el cual hemos llamado LindaServer, que gestiona varios procesos cliente, y que a su vez trata 3 procesos servidores.

Los clientes disponen de 5 operaciones: postnote, removenote, readnote, conectarse al servicio o desconectarse. Los servidores (1, 2 y 3) son los responsables de almacenar y tratar las operaciones que involucran a las tuplas enviadas por los clientes. Cada servidor gestiona el almacenamiento de tuplas de unas determinadas dimensiones en su memoria. En concreto, un servidor trabaja con tuplas entre 1 y 3 elementos, otro entre 4 y 5 y otro de solo 6 elementos.

Para este trabajo las tuplas tendrán una longitud máxima de 6 elementos (por simplicidad). También existen patrones, son tuplas, pero uno o más de sus elementos son variables a las cuales se les pueden ligar otros valores. Los patrones podrán ser solo usados por las operaciones removenote y readnote.

1.2 Participantes del proyecto

Los componentes del equipo son:

- Germán Garcés Latre
- Diego Caballé Casanova
- Sergio García Esteban
- Patricia Briones Yus

1.3 Estructura

Esta es la lista de los módulos que hemos desarrollado a lo largo del trabajo:

- **Tuplas.cpp** y **Tuplas.hpp**. Implementación del TAD Tupla, que está formado por el número de elementos que contiene, y un vector de strings donde se guarda la información de cada elemento.
- **LindaServer.cpp**. Crea un servidor que atiende a varios clientes, y envía a los servidores el cometido de cada uno.
- **Servidor.cpp**. Crea un servidor que según la operación a realizar enviada desde LindaServer insertará, eliminará o leerá una tupla de memoria.

- **LindaDriver.cpp** y **LindaDriver.hpp**. Monitor que se encarga de la conexión de los clientes con LindaServer y a su vez de las operaciones a las que tienen acceso los clientes.

- **ControlLS.cpp** y **ControlLS.hpp**. Monitor que se encarga de controlar todo lo que pasa dentro de LindaServer en exclusión mutua.

- **Admin.cpp**. Crea un cliente que se conecta al servicio cuya función es cerrar el LindaServer de manera controlada y como proceso administrador.

- **ControlTuplas.cpp** y **ControlTuplas.hpp**. Monitor que se encarga de insertar, eliminar o leer las tuplas de la memoria en exclusión mutua.

- **Clienteprueba.cpp**. Crea un cliente que se comunica conecta al servicio para realizar distintas operaciones.

2. Diseño de alto nivel de la solución

El sistema está compuesto por: tres procesos servidor, un proceso “LindaServer” (servidor y cliente) y uno o varios procesos cliente.

Los clientes pueden optar por 5 opciones: postnote, removenote, readnote, conectarse al servicio o desconectarse. LindaDriver se encarga de transmitir a LindaServer, a través de la red, las operaciones a realizar por los clientes.

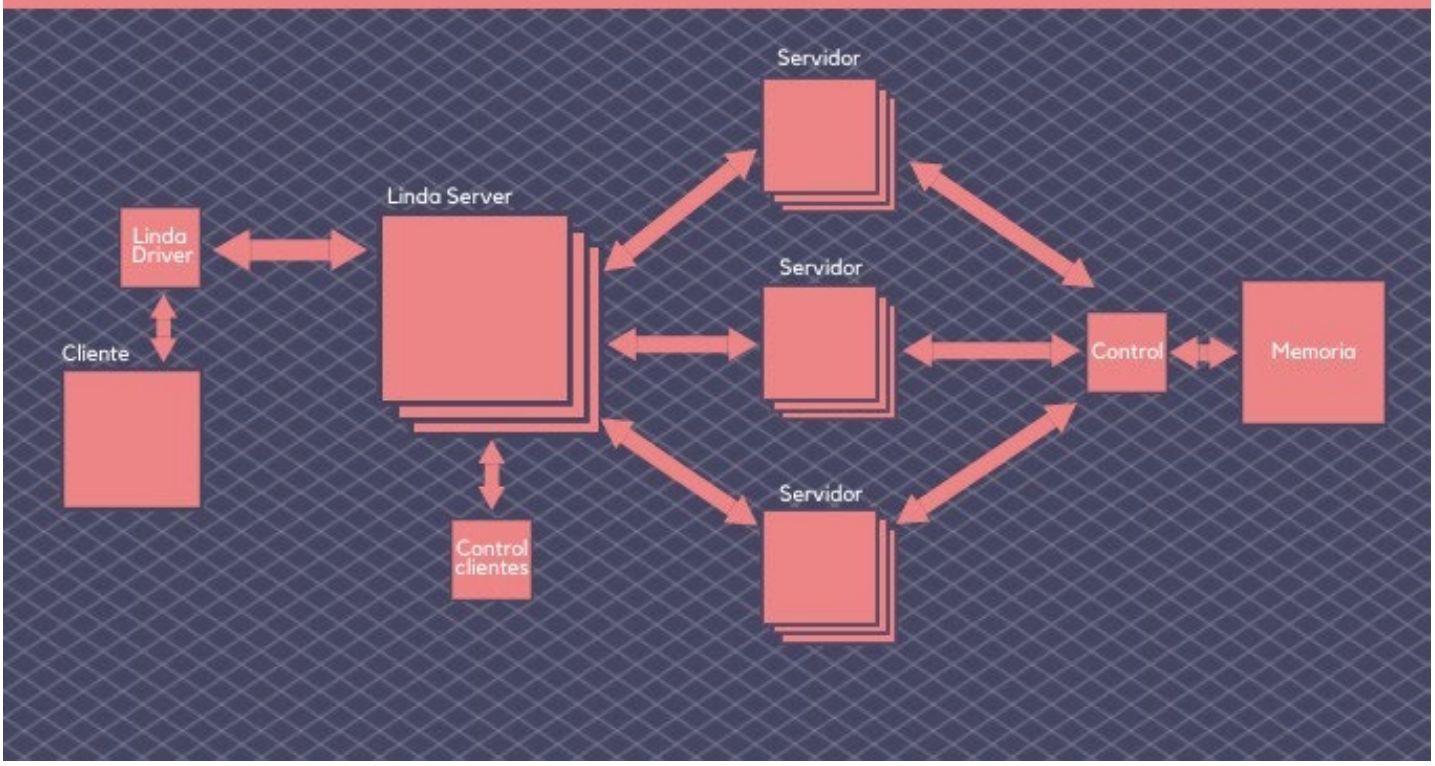
Para conectarse al servicio, los clientes conocen el puerto por el que se comunican con LindaServer, y se crea una red de comunicación entre ambos a través del monitor LindaDriver. LindaServer lanzará un thread por cada cliente conectado y recibirá a continuación una de las otras 4 opciones (no contamos conectarse al servicio) que haya escogido el cliente a través de LindaDriver.

LindaServer está conectado por red a los servidores 1, 2 y 3 (cuyos puertos se asignan en la invocación de LindaServer), actuando en este caso como un cliente y no como servidor. Cuando LindaServer se conecta a uno de los servidores, estos crean un thread, después recibe el mensaje, se reconoce la acción a realizar y crea la tupla. Cada servidor controla su memoria con un monitor “ControlTuplas” que realiza las operaciones en exclusión mutua.

Para apagar LindaServer de manera controlada, hemos creado un cliente “administrador” que se conecta a LindaServer y envía a través de la red el mensaje “FIN_SERVER". LindaServer se conectará a los tres servidores que manejan la memoria de tuplas y les enviará el mensaje “TERMINACIÓN” a cada uno de ellos. Cuando los servidores detecten ese mensaje, cerrarán la conexión con LindaServer. En caso de estar un cliente bloqueado esperando una tupla, se despertarán los procesos del monitor ControlTuplas, saldrán del bloqueo y enviarán una tupla vacía para, de ese modo, acabar.

LINDA

Representación gráfica



3. Decisiones de diseño relevantes

3.1 Monitor LindaDriver

La función principal de LindaDriver es crear una comunicación de red entre el cliente y el servidor de Linda (LindaServer). También ofrece otras 4 operaciones distintas:

- **Desconectarse del servicio**. LindaDriver envía a LindaServer el mensaje "DESCONECTAR CLIENTE" y cierra por tanto la conexión entre ambos cerrando el socket, desconectando de este modo al cliente del servicio.

- **PostNote, RemoveNote, ReadNote**. LindaDriver envía a LindaServer el mensaje con formato "INSTRUCCIÓN[INFO_TUPLA]NºELEM_TUPLA". Si el número de elementos de la tupla es entre 1 y 3, LindaServer creará la conexión con el servidor 1 y le enviará el mensaje recibido anteriormente, si el número de elementos es entre 4 y 5 hará lo mismo para el servidor 2, y si el número de elementos es 6, hará lo mismo para el servidor 3. Después espera el mensaje de LindaServer del resultado de la operación.

3.2 Monitor ControlLS

Para llevar el control del número de clientes que hay conectados a la vez al servicio y el número de tuplas que hay en la pizarra en todo momento, LindaServer usa un monitor "ControlLS" en exclusión mutua.

Está formado por: el número de procesos cliente que están conectados, el número de tuplas que hay en la pizarra, un booleano fin para saber si hay que cerrar la entrada a los clientes, un booleano salir para saber si hay que sacar a los clientes del servidor y una condición mainEspera para bloquear el main. El monitor consta de 6 operaciones:

- **EntraCliente** / **SaleCliente**: cada vez que un cliente entra (se conecta) o sale de LindaServer se incrementa o decrementa el número de clientes conectados.

- **ActualizaTuplas**: después de una instrucción, actualiza el valor de las tuplas en la pizarra.

- **Finalización**: cambia el valor de la variable fin a verdadero.

- **Salida**: devuelve el valor de salir.

- **Acabo**: devuelve el valor de fin.

3.3 Monitor ControlTuplas

El monitor ControlTuplas se encarga de realizar las operaciones de insertar, eliminar y leer tuplas de la memoria de cada servidor en exclusión mutua.

Está formado por: una estructura de datos de tipo lista donde las tuplas serán almacenadas (la cual fue implementada por nosotros), un booleano salir para saber cuándo el servidor se quiere cerrar y una condición esperandoTupla para bloquear las operaciones eliminar y leer en caso de que la tupla no exista. El monitor consta de 4 operaciones:

- **Añadir**. Se asegura de que la tupla no es un patrón, la introduce, despierta al resto de procesos por si hay alguno bloqueado y termina. Dando paso a que el servidor envíe a LindaServer una confirmación de que ha sido introducida correctamente. Si lo que se intenta introducir es un patrón, habrá error y el servidor enviará un mensaje de que la inserción no se ha podido realizar.

- **Leer**. Comprueba si la tupla que se quiere leer no está en memoria. En caso de que no, se bloquea hasta que esa tupla exista en memoria. Una vez ha sido comprobado se actualiza el valor de sus elementos en caso de que fuese un patrón. También se comprueba si el servidor se quiere cerrar.

- **Eliminar**. Mismas operaciones que ReadNote, pero una vez actualizados los valores de los elementos de la tupla, la elimina de memoria.

- **Finalizar**. Cambia el valor de salir a verdadero y despierta a todos los procesos que hubiese bloqueados para indicarles que tienen que salir.

3.4 TAD Tuplas

El TAD consta de varias operaciones. Como variables permanentes tiene tamaño que indica el número de elementos de la tupla y cadena que es donde se almacenan los datos de los elementos. Principalmente, cuenta con 7 constructores diferentes, de los cuales 5 de ellos son introduciendo los elementos separados por comas (con un máximo de 6 elementos) y otro que solo necesita de un número entre 1 y 6, que será el número de elementos de los que se compondrá la tupla, y todos sus campos en blanco.

Las otras 7 operaciones restantes (se nos había informado de muchas de ellas en el trabajo) son:

- **Set**. Establece el valor del elemento en el apartado n de la tupla.

- **Get**. Devuelve el string correspondiente al apartado n de la tupla.

- **To_string**. Devuelve todos los elementos de la tupla en formato [elemento, …].

- **From_string**. Modifica el contenido de la tupla para que quede como la información introducida. La información tiene que contener el mismo número de elementos de la tupla a cambiar.

- **Size**. Devuelve el número de elementos de la tupla.

- **EsVariable**. Devuelve true si el elemento de la tupla indicado es una variable.

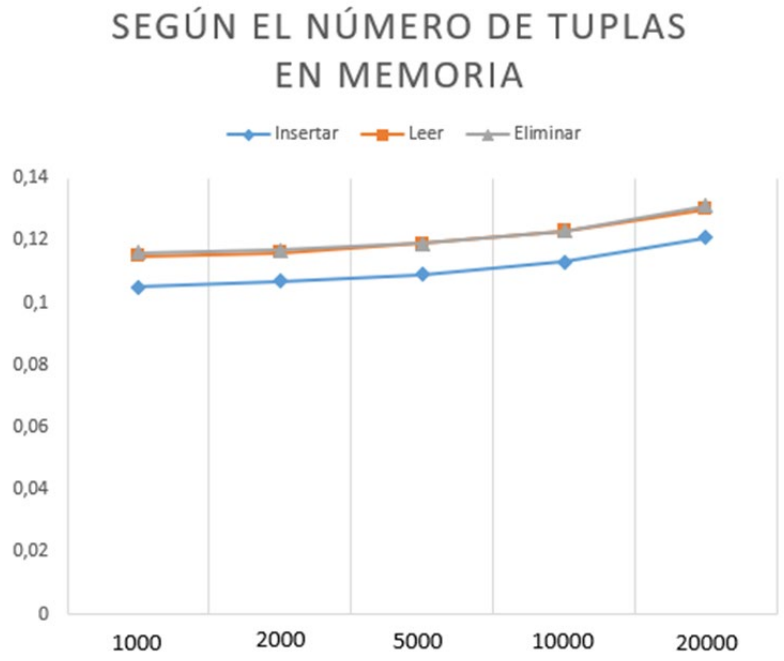
- **Letra**. Devuelve la posición de una letra en el abecedario.

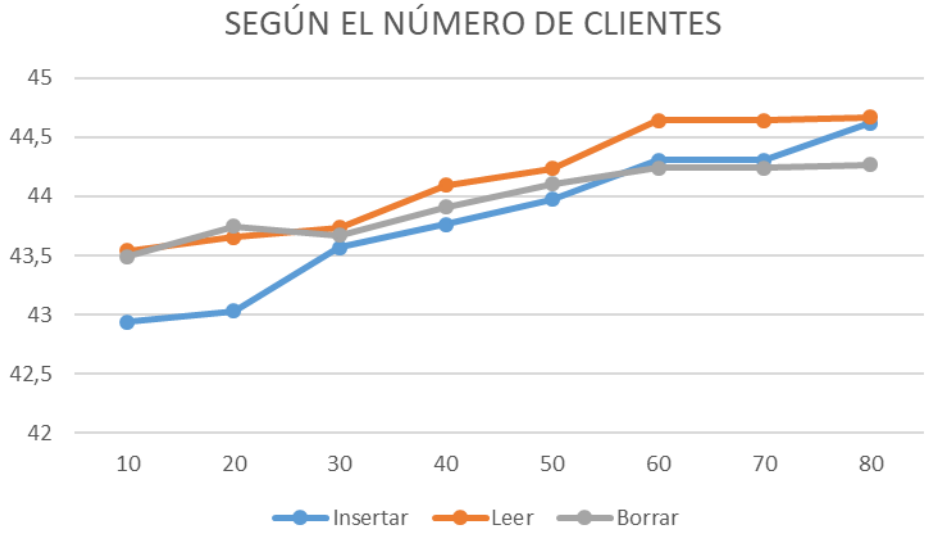
4. Evaluación del sistema final

Para ejecutar nuestro sistema Linda hacemos uso de 1 máquina. El servidor LindaServer necesita un puerto de la misma máquina, hemos estado usando el 5000, al que el cliente se conecta, los servidores, cada uno en un puerto diferente han estado usando los puertos 5001, 5002, 5003 para el servidor uno, dos y tres respectivamente.

El hardware usado ha sido diferente en ciertos momentos, desde una i7 con sistema operativo Arch a una i5 con sistema operativo Ubuntu, pero siempre hemos usado una máquina que controlaba todos los procesos a la vez, incluso en las pruebas en Hendrix.

Experimentos de evaluación del tiempo de ejecución de las instrucciones según:





5. Planificación y organización del trabajo

Conjuntamente se quedó para leer el trabajo, entenderlo y realizar un primer diseño de lo que nos pedían. El diseño no siempre ha sido el mismo, ya que a lo largo del proyecto se ha ido cambiando en base a lo que íbamos creyendo que era mejor.

Respecto a la distribución del tiempo, nos organizamos bastante bien, cumpliendo nuestros propios plazos en cuanto a tener el código hecho de un determinado fichero "X" o solución a algún problema.

5.1 Tareas desarrolladas

Aunque todos nos hemos ayudado los unos a los otros, el trabajo se ha gestionado principalmente de la siguiente manera:

Germán Garcés Latre

Creación de los objetos a usar en el trabajo: como las tuplas, o la manera de organizarlas en memoria.

Sergio García Esteban

Diseño de la interacción cliente lindaServer y servidores y de los programas involucrados. Realización de las pruebas de LindaDriver, LindaServer y posteriormente de servidores. Debuggeo del programa en la unión de las tres partes.

Diego Caballé Casanova

Diseño de la interacción cliente, LindaServer y servidores y de los programas involucrados. Creación de LindaDriver, LindaServer , controlLS y proceso admin. Debuggeo del programa en la unión de las tres partes. Desarrollo de la terminación de los programas LindaServer y LindaDriver así como las pruebas de su terminación.

Patricia Briones Yus

Desarrollo de servidores y controlTuplas. Realización de las pruebas de los servidores, creación de toda la documentación referida a la entrega, implementación de las terminaciones de servidores y los patrones.

5.2 Control de esfuerzos

El tiempo invertido en el proyecto por cada uno de nosotros ha sido:

- Germán Garcés Latre: 20h 15min
- Sergio García Esteban: 33h
- Diego Caballé Casanova: 56h 30 min
- Patricia Briones Yus: 42h

6. Conclusiones y posibles mejoras futuras

Una de las posibles mejoras que nos hemos planteado, es el uso de estructuras AVL en vez de listas para el almacenamiento de las tuplas y así no tener un coste de tiempo lineal, si no logarítmico a la hora de realizar las operaciones. Hicimos un intento de hacer uso del AVL, pero nos causaba tantos problemas que decidimos cambiarlo y no perder más de nuestro valioso tiempo.

Lo más acertado ha sido el diseño del sistema, debido a que se le dedicó una gran parte del tiempo para más tarde facilitarnos la implementación del código.