Trabajo no presencial: Implementación de un Linda distribuido

Programación de Sistemas Concurrentes y Distribuidos
Grado de Ingeniería Informática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

3 de diciembre de 2020

1. Objetivos

Los objetivos de este trabajo no presencial son los siguientes:

- Profundizar en los modelos de comunicación síncrona/asíncrona para procesos distribuidos.
- Asentar los conocimientos adquiridos para la programación de aplicaciones distribuidas en C++.
- Desarrollar una aplicación distribuida cliente/servidor.
- Trabajar en equipo.

2. Descripción del sistema de coordinación

El objetivo del trabajo es desarrollar un sistema de coordinación Linda distribuido. El sistema ofrecerá las operaciones básicas vistas en clase, conforme a la semántica presentada. Estas operaciones son: `PN`(postNote), `RN` (removeNote) y `RdN` (readNote). Por simplicidad, las tuplas serán **planas**, es decir, no existen tuplas donde uno de sus elementos sea a su vez otra tupla (tuplas anidadas), tendrán una **longitud máxima de 6** elementos y todos los elementos serán de tipo **string**.

Un patrón será un tipo especial de tupla en el que pueden aparecer *variables*, que podrán, mediante una operación `RN` o `RdN`, ligarse a valores. Así [“*Juan*”, “45”, “34”, “88”] es una tupla/patrón de 4 elementos concretos, mientras que [“*Juan*”, “?*X*”, “34”, “?*Y*”] es un patrón con dos variables. Por simplicidad, una variable en un patrón se indicará mediante el string compuesto por “?” y una letra mayúscula entre “A” y “Z”.

Además de las operaciones básicas, el sistema Linda distribuido ofrecerá también las operaciones `RN_2` (removeNote múltiple) y `RdN_2` (readNote múltiple). La semántica de estas operaciones es la siguiente. Cada operación de lectura consta de dos tuplas patrón, que pueden contener variables propias y/o comunes.

Por tanto, a diferencia de las operaciones básicas, el objetivo de estas operaciones es leer dos tuplas "de golpe" conforme a los patrones especificados como entrada; si no has hubiera, las operaciones de lectura se bloquean hasta que ambas tuplas estén disponibles. Para comprender cómo funcionan las variables en estas lecturas múltiples, supongamos que se especifican los patrones de entrada ([“$Juan$”, “?X”, “34”, “?Y”], [“$Luis$”, “76”, “?X”, “12”]). Estos tienen una variable común X y, además, el primer patrón tiene una variable propia Y. Las variables comunes deben ligarse a un único valor, es decir, cuando la operación de lectura se complete las tuplas leídas deben cumplir que el valor del segundo elemento de la primera tupla sea el mismo que el valor del tercer elemento de la segunda. Por otro lado, la semántica de las variables propias (la variable Y) es la misma que en las operaciones básicas de Linda.

La figura 1 muestra una abstracción del sistema a desarrollar. Los *procesos C++* actúan como clientes del sistema de coordinación Linda. Antes de interaccionar con Linda deben descubrir dónde está desplegado el sistema de coordinación (básicamente, es necesario conocer la dirección IP y el puerto del servidor Linda). Para ello se comunican con un servidor que actúa como un *Registro de despliegue* y que conoce esta información. El protocolo de comunicación concreto entre los procesos y el registro deberá especificarse como parte del trabajo. Una vez se conozca la ubicación del servidor de coordinación, un *proceso C++* usará la librería *Linda driver*, que también debe implementarse como parte de la solución, para invocar remotamente las operaciones de lectura/escritura de tuplas.

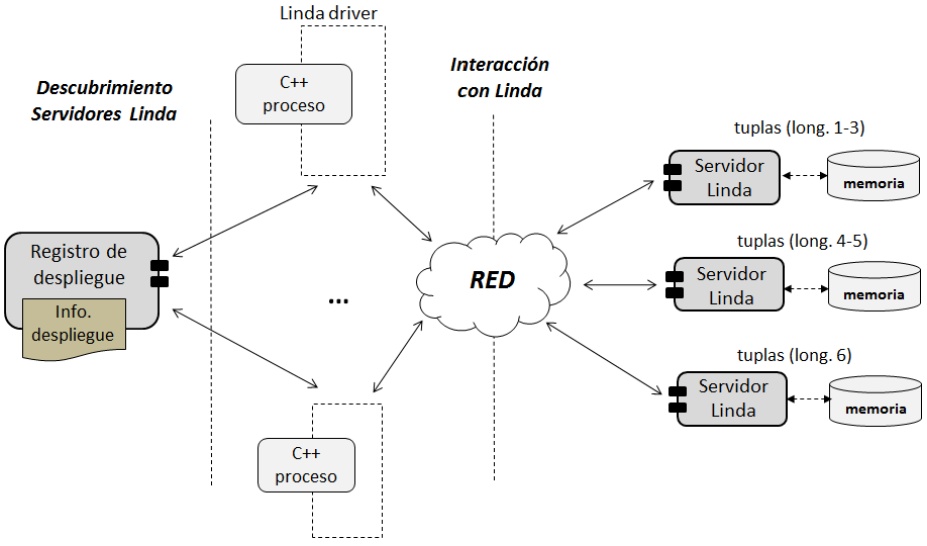

Figura 1: Esquema del sistema

Por otro lado, el sistema de coordinación Linda (parte derecha de la figura) estará compuesto por *tres servidores*, posiblemente ejecutándose en diferentes máquinas, responsables de almacenar y gestionar las operaciones que involucran a las tuplas que cada uno de ellos almacena. En concreto, el primer servidor trabajará con las tuplas de tamaño 1 a 3, el segundo con las de tamaño 4 y 5,

mientras que el tercero se encargará de las tuplas de tamaño 6. Internamente, las tuplas se almacenarán en memoria, delegando en los equipos de desarrollo la elección de las estructuras de datos que consideren más adecuadas.

El Anexo-I muestra un ejemplo de uso del entorno, tanto desde el punto de vista de la construcción y del manejo de tuplas y patrones como del uso del servidor Linda.

3. Un sistema de gestión de ofertas de viaje

Una vez programado el sistema Linda descrito en la sección anterior, el siguiente objetivo es aplicarlo a un problema concreto, resolviendo las cuestiones de coordinación involucradas. Este problema consiste en facilitar bajo ciertas condiciones el intercambio distribuido de información relativa a ofertas de viaje.

Un sistema va publicando ofertas de viaje. Cada oferta consiste de la ciudad de origen, la de destino y el precio del viaje, por persona. Por simplicidad podemos suponer que el sistema trabaja con la información relativa a una región geográfica compuesta por 10 ciudades cualesquiera.

El sistema consta de tres tipos de procesos:

- los procesos *publicador* son responsables de escribir nuevas ofertas en el sistema, representando el orden de publicación por medio de un valor numérico que será incrementado y añadido a cada oferta concreta (establece una relación de orden global de las ofertas, empezando con el valor 1 cuando se inicia el sistema).
- los procesos *buscador* leen un número aleatorio de entre 1 y 3 ofertas que fueron publicadas de forma consecutiva en el sistema y escriben por la salida estándar (pantalla) la información de las mismas. Repiten este comportamiento durante 10 veces antes de finalizar su actividad. Cada una de estas repeticiones se considera un acceso al sistema.
- y los procesos *buscador_combinados* buscan ofertas que permitan viajar de una ciudad origen dada a otra destino también dada, pasando por una tercera ciudad intermedia. Las ciudades origen y destino serán elegidas en base a algún criterio entre las disponibles (aleatorio, por ejemplo). Repite este comportamiento durante 5 veces, considerando cada repetición un acceso al sistema.

El sistema está integrado por 5 publicadores, 10 buscadores simples y 5 buscadores de ofertas combinadas. Además, se ha establecido la restricción de que **en todo momento, como máximo, puede haber 8 procesos buscando ofertas** (simples o combinados). Por este motivo, cada vez que un buscador quiera acceder al sistema deberá obtener permiso y esperar, si fuera necesario.

Se pide programar los distintos procesos involucrados en el sistema y coordinar su actividad a través de Linda. Como parte de la solución será necesario especificar los protocolos de comunicación entre los procesos y el sistema de coordinación, así como las tuplas intercambiadas como parte de esos protocolos. El sistema final debe ser fácilmente ejecutable en un entorno distribuido y, por tanto, se deberá también programar todos aquellos procesos auxiliares que faciliten la puesta en marcha de la solución.

4. Criterios de puntuación y funcionalidades extra

Aquellos equipos que programen el sistema distribuido Linda y resuelvan el problema de la gestión de ofertas de viaje correctamente podrán obtener una nota máxima de 8 puntos sobre 10. Para optar a la nota máxima, 10 puntos, deberán programarse también las siguientes mejoras:

- En primer lugar, se pide añadir al sistema distribuido Linda un servidor de *monitorización* que ofrezca periódicamente por las salida estándar información estadística sobre el estado del sistema de coordinación (número de tuplas actualmente en los repositorios, número de peticiones de lectura/escritura recibidas y/o pendientes, etc.). Este servidor deberá programarse de forma que se pueda ejecutar en una máquina independiente a las utilizadas por los tres servidores Linda. Como parte del trabajo se deberá definir el protocolo de comunicación entre los servidores involucrados, las métricas a controlar, y la estrategia de actualización de dichas métricas.
- Dada las características del sistema de gestión de ofertas de viaje, es posible que algunos procesos no acaben su ejecución, en función del número de ofertas que se publiquen y/o consuman. Se pide diseñar y programar una estrategia de finalización controlada que garantice que, cuando se cumplan ciertas condiciones razonables y debidamente justificadas, todos los procesos del sistema concluirán su ejecución.

5. Instrucciones para el desarrollo y la entrega del trabajo

Es importante que os leáis con detenimiento esta sección. Incluye una panorámica general de las actividades e instrucciones a tener en cuenta para el desarrollo del proyecto e información sobre los hitos y fechas de interés.

5.1. Organización inicial del trabajo

La realización de este proyecto es una actividad en grupo. En el curso *Moodle* de la asignatura se publicaron las instrucciones y los plazos para la creación de los equipos de trabajo. Para poder entregar y defender el proyecto realizado es obligatorio haber completado este trámite previo.

5.2. Material a entregar como resultado

Esta primera versión del enunciado se centra en describir el sistema objetivo a desarrollar. En próximas fechas se publicará una versión complementaria donde se describa con precisión todo el material a entregar como resultado del proyecto. También se publicarán en el curso *Moodle* de la asignatura una serie de plantillas para elaborar la documentación final del proyecto.

A nivel de resumen, queremos anticipar que cada equipo deberá entregar como mínimo el siguiente material:

- El código fuente del sistema desarrollado y los correspondientes ficheros `Makefile` para su compilación y ejecución.
- La memoria técnica del proyecto, donde se describa el análisis y diseño de la solución implementada. Esta memoria también contendrá un análisis de rendimiento y prestaciones del sistema de coordinación Linda.
- La memoria organizativa del proyecto, donde se detalle la organización del equipo, el plan de trabajo planificado/real y los esfuerzos dedicados al proyecto.
- Un documento de pruebas que recoja el plan de pruebas realizado para garantizar el correcto funcionamiento del sistema.
- La presentación que se va a realizar para la defensa del proyecto.

En el curso *Moodle* de la asignatura se publicarán instrucciones precisas que guíen a los equipos en la elaboración de los distintos documentos y las correspondientes plantillas.

5.3. Gestión de la dedicación

Como parte de la documentación a entregar se pide un informe de dedicación al proyecto. Este informe debe detallar las horas que cada estudiante ha dedicado a las distintas tareas del proyecto (participación en reuniones, programación, realización de pruebas, elaboración de la documentación, etc.). Esta información no tiene ningún tipo de consecuencia a nivel de evaluación y calificación, simplemente es una práctica habitual en el desarrollo de proyectos *software*. El objetivo es conocer el esfuerzo que ha costado a los estudiantes realizar el trabajo y valorar objetivamente si corresponde con el planificado en la ficha de la asignatura. Por tanto, solicitamos sinceridad en los datos que se entreguen. Nos ayudará a mejorar la asignatura.

Para facilitaros la labor os proporcionamos un documento `Excel` para la gestión de los esfuerzos individuales. Cada vez que trabajéis en el proyecto debéis apuntar las horas dedicadas, especificando la fecha y hora, la tarea realizada, y el tiempo invertido en fracciones de 15 minutos (es decir, si dedico 12 minutos, apuntaría 15 minutos de trabajo; o si dedico 37 minutos, apuntaría 45 minutos de trabajo). En el documento también se proporciona un listado de tareas posibles.

Por otro lado, está la cuestión de cómo computar las horas que se trabaja en equipo. Si por ejemplo se reúnen tres estudiantes para realizar una tarea durante 50 minutos, cada estudiante deberá apuntar en su dedicación individual que participó en la tarea durante 1 hora. Es decir, el esfuerzo computable es 50 minutos por el número de participantes. Esta regla se aplicará a cualquier tarea en grupo.

Finalmente, queremos recordaros que debéis ser disciplinados a la hora de registrar la información de esfuerzos. Una buena costumbre es apuntar la información nada más finalizar cada tarea.

5.4. Procedimiento y fechas de entrega del proyecto

La fecha límite para la entrega del material del proyecto será el día 13 de enero del 2021, a las 23:59h. Las instrucciones y condiciones de entrega serán

publicadas con suficiente antelación. Esta entrega supone el primer paso en el proceso de evaluación de la actividad.

El segundo paso consiste en la evaluación presencial de los proyectos. Esta evaluación incluye una breve presentación de los resultados y una demostración en el laboratorio de prácticas del sistema implementado. Una vez entregados los trabajos, se publicarán con antelación suficiente las fechas y horas concretas en las que se realizará esta evaluación presencial. La asistencia a esta prueba es obligatoria para todos los integrantes del equipo. En caso contrario, el equipo será calificado como *no presentado* o, incluso, *suspenso*, según las circunstancias. La fecha de la defensa será los días 18 y 19 de enero del 2021 (aunque estas fechas están por confirmar). También se publicarán instrucciones sobre el desarrollo de esta prueba.

Anexo-I: Ejemplo de uso de Linda desde el punto de vista de un programa cliente

```
1  #include "LindaDriver.hpp"
2  #include "Tupla.hpp"
3  ...
4
5  using namespace std;
6
7  //-----------------------------------------------------
8  int main(int argc, char* argv[]) {
9
10     //faltan argumentos en la invocación?
11     if (argc < ... ) {
12         cerr << "Invocar como:" << endl
13              << "   mainLindaDriver <IP_LS> <Port_LS> ..." << endl
14              << "       <IP_LS>: IP del servidor Linda" << endl
15              << "       <Port_LS>: puerto del servidor Linda" << endl;
16         return 1;
17     }
18
19     // un driver con los datos pasados en la invocación
20     LindaDriver LD(argv[1], argv[2], ...);
21     // Ya ha contactado con el registro, obtenido los datos necesarios de
22     // los tres servidores y establecido las conexiones requeridas
23
24     // Varias formas de construir tuplas
25     Tupla t1("1","mi casa","árbol"); // 3 elementos
26     Tupla t2("1000");
27     Tupla t3("aprieta","el","pan","45","34","88");
28     Tupla t4("aprieta","fuerte","pan","tt","34","pan");
29     // insertar las tuplas en linda: varios PostNote
30     LD.PN(t1);
31     LD.PN(t2);
32     LD.PN(t3);
33     LD.PN(t3);
34     LD.PN(t3);
35     LD.PN(t4);
36
37     //muestra "mi casa" por stdout
38     cout << t1.get(2) << endl;
39     // t3.to_string() devuelve el string "[aprieta,el,pan,45,34,88]"
40     string serial = t3.to_string();
41     cout << serial << endl;
42     // las componentes de t3 tomarán, respectivamente,
43     // los valores "a","b","c","45","34","pan"
44     t3.from_string("[a,b,c,45,34,pan]");
45     // mostrará [a,b,c,45,34,pan] por stdout
46     cout << t3.to_string() << endl;
47     ...
48
49     // Crea una tupla de 3 elementos "".
50     // Equivalente a Tupla t5("","","")
```

```
51      Tupla t5(3);
52      t5.set(2, "hola");
53      t5.set(3, "Mundo");
54      LD.PN(t5);
55      // mostrará [,hola,Mundo] por stdout
56      cout << t5.to_string() << endl;
57      // informará de que tiene 3 elementos
58      cout << "t5␣tiene␣" << t5.size() << "␣elementos" << endl;
59      ...
60
61      // Un patrón no es otra cosa que una tupla con
62      // la posibilidad de contener el comodín "?" en una o más posiciones
63      // "?" no puede ser un valor de componente de tupla. Para ello se
64      // tendría que representar mediante "??". Creamos dos patrones
65      Tupla p1("?X");
66      Tupla p2("a","?X","c","?Y","34","?Z");
67      // Dos nuevas tuplas, de tamaño 1 y 6, respectivamente
68      Tupla res1(1),
69            res2(p2.size());
70      // ejemplos de RemoveNote
71      LD.RN(p1, res1); // res1 tomará él valor que tenía t2
72      LD.RN(p2, res2); // res2 tomará el valor que tenía t3
73      cout << res1.to_string() << endl; //mostrará [1000]
74      cout << res2.to_string() << endl; //mostrará [a,b,c,45,34,pan]
75      ...
76
77      // ¿Si necesitamos un array de tuplas?
78      // Tupla v[2]; // NO permitido: no hay constructor por defecto
79      Tupla* v[2];
80      v[0] = new Tupla("Juan", "1000");
81      v[1] = new Tupla("Luisa", "1000", "enero");
82      ...
83      delete v[0];
84      delete v[1];
85
86      Tupla t6("EINA","AOC","DIIS");
87      Tupla t7("EINA","PSCD","DIIS");
88      LD.PN(t6);
89      LD.PN(t7);
90      Tupla p3("EINA","?Y","?X");
91      Tupla p4("EINA","?Z","?X");
92
93      Tupla t8(3), t9(3);
94      LD.RN_2(p3,p4,t8,t9);
95      // Podrá haber cargado t8 con ["EINA","AOC","DIIS"] y
96      // t9 con ["EINA","PSCD","DIIS"], o viceversa
97      cout << t8.to_string() << endl;
98      cout << t9.to_string() << endl;
99  }
```

Listado 1: Ejemplo de uso de linda