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

4 de diciembre de 2018

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 vistas en clase, y con la misma semántica introducida en ellas. Las operaciones serán: `PN`, `RN` y `readN`. 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 `readN`, ligarse a valores. Así ["*Juan*", "45", "34", "88"] es una tupla/patrón de 4 elementos, 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".

La figura 1 muestra una abstracción del sistema a desarrollar. El *servidor Linda* publica la interfaz del sistema de coordinación distribuido. Esta interfaz ofrece a los procesos remotos (parte izquierda de la figura) cinco operaciones: conectarse al servicio y desconectarse de él, además de las tres operaciones de manipulación de tuplas. Un *proceso C++* usará la librería *Linda driver*, que

debe implementarse como parte de la solución, para invocar estas operaciones remotamente.

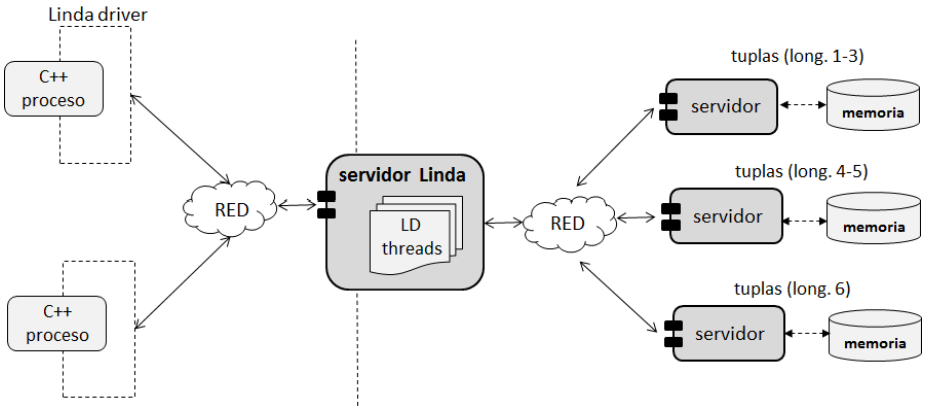

Figura 1: Esquema del sistema

Por otro lado, el *servidor Linda* estará compuesto por *tres servidores*, posiblemente ejecutándose en diferentes máquinas, responsables de almacenar y gestionar las operaciones que involucran a las truplas que ellos almacenan. 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-4 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. 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.

3.1. Organización inicial del trabajo

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

3.2. 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 la página Web una serie de plantillas para elaborar la documentación final del proyecto.

A nivel de resumen, queremos anticipar que cada grupo 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 final.
- 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 la página Web se publicarán instrucciones precisas que guíen a los grupos en la elaboración de los distintos documentos y las correspondientes plantillas.

3.3. 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.

3.4. Procedimiento y fechas de entrega del proyecto

La fecha límite para la entrega del material del proyecto será el día 14 de enero del 2019, 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 grupo. En caso contrario, el grupo será calificado como *no presentado* o, incluso, *suspenso*, según las circunstancias. La fecha de la defensa será los días 17 y 18 de enero del 2019. También se publicarán instrucciones sobre el desarrollo de esta prueba.

4. Anexo: Ejemplo de uso de Linda desde el punto de vista de un programa cliente

```
#include "LindaDriver.hpp"
#include "Tuplas.hpp"
using namespace std;

int main(int argc, char* argv[]) {

    //faltan argumentos en la invocación?
    if (argc < ... ) {
        cerr << "Invocar como:" << endl;
        cerr << "   mainLindaDriver <IP_LS> <Port_LS> ..." << endl;
        cerr << "          <IP_LS>: IP del servidor Linda" << endl;
        cerr << "          <Port_LS>: puerto del servidor Linda" << endl;
        return 1;
    }

    // un driver con los datos pasados en la invocación
    LindaDriver LD(argv[1], argv[2],  ...);
    // La conexión con el servidor Linda ya está establecido

    // Varias formas de construir tuplas
    Tupla t1("1","mi casa","árbol");  // 3 elementos
    Tupla t2("1000");
    Tupla t3("aprieta","el","pan","45","34","88");
    Tupla t4("aprieta","fuerte","pan","tt","34","pan");
    // insertar las tuplas en linda: varios PostNote
    LD.PN(t1);
    LD.PN(t2);
    LD.PN(t3);
    LD.PN(t3);
    LD.PN(t3);
    LD.PN(t4);

    //muestra "mi casa" por stdout
    cout << t1.get(2) << endl;
    // t3. to_string () devuelve el string "[aprieta, el, pan,45,34,88]"
    string serial = t3.to_string ();
    cout << serial << endl;
    // las componentes de t3 tomarán, respectivamente,
    // los valores "a","b","c","45","34","pan"
    t3.from_string("[a,b,c,45,34, pan]");
    // mostrará [a,b,c,45,34,pan] por stdout
    cout << t3.to_string() << endl;
    ...

    // Crea una tupla de 3 strings "".
    // Equivalente a 'Tupla t5("","","")'
    Tupla t5(3);
    t5.set (2, "hola");
    t5.set (3, "Mundo");
    LD.PN(t5);
```

```
    // mostrará [,hola,Mundo] por stdout
    cout << t5.to_string() << endl;
    // informará de que tiene 3 elementos
    cout << "t5_tiene_" << t5.size() << "_elementos" << endl;
    ...

    // Un patrón no es otra cosa que una tupla con
    // la posibilidad de contener variables "?X" en una o más posiciones
    // Creamos dos patrones
    Tupla p1("?X");
    Tupla p2("a","?X","c","?Y","34","?Z");
    // Dos nuevas tuplas, de tamaño 1 y 6, respectivamente
    Tupla res1(1),
          res2(p2.size());
    // ejemplos de RemoveNote
    res1 = LD.RN(p1); // res1 tomará él valor que tenía t2
    res2 = LD.RN(p2); // res2 tomará el valor que tenía t3
    cout << res1.to_string() << endl; //mostrará [1000]
    cout << res2.to_string() << endl; //mostrará [a,b,c,45,34,pan]
    ...

    // ¿Si necesitamos un array de tuplas?
    // Tupla v[2]; NO permitido: no hay constructor por defecto
    Tupla *v[2];
    v[0] = new Tupla("Juan", "1000");
    v[1] = new Tupla("Luisa", "1000", "enero");
    ...
    delete v[0];
    delete v[1];

    return 0;
}
```