Telecomunicaciones y Sistemas Distribuidos

Proyecto de aprobación

2014

Resumen

Este proyecto pretende que el estudiante aplique algunos de los algoritmos distribuidos analizados en el curso. Además requiere que se utilicen los conceptos sobre comunicaciones, diseño e implementación de protocolos de comunicaciones.

1. El problema

Se requiere desarrollar un sistema distribuido en el que se simule un conjunto de procesos ejecutándose en forma concurrente o paralela en un sistema de computación con un mecanismo de pasaje de mensajes asíncrono.

La venta de pasajes de ómnibus o (cualquier otro tipo de medio de transporte) en un recorrido con escalas presenta dificultades ya que en cada punto de venta (terminal) del recorrido se pueden realizar reservas, ventas o devoluciones (cancelaciones).

Muchos sistemas de este tipo solucionan este problema mediante un sistema centralizado (cliente-servidor), donde el servidor mantiene el estado global del sistema (el vehículo de transporte) y los clientes realizan las operaciones de consulta, reserva (o venta) y cancelación (devolución) de lugares realizando transacciones con el servidor, utilizando algún protocolo de comunicación o middleware como remote-procedure-call (RPC).

La solución centralizada requiere que los clientes estén on-line permanentemente, concentra las comunicaciones (requiriendo un ancho de banda superior para el servidor) y ofrece un único punto de falla.

Una solución distribuida plantea varios inconvenientes como la recolección del estado global y la consistencia (mantener las invariantes) en las operaciones.

2. La solución

Un modelo abstracto orientado a objetos con contratos del vehículo (secuencial) puede describirse como:

```
vehicle {

  init() {
    reserved = 0;
  }

  // return available seats
  int available() {
    ensure(result == seats - reserved);
  }

  // reserve n seats
  bool reserve(int n)
  {
    require(n <= available());
    ensure(reserved == old_reserved + n);
  }

  // cancel n seats
  bool cancel(int n)
  {
    require(n <= reserved);
    ensure(reserved == old_reserved-n);
  }

  const int seats = N;
  int reserved;
};
```

Se requiere implementar el sistema distribuido para el problema descripto sin usar una coordinación central. Podrá usar las herramientas de desarrollo de su elección.

Para simplificar el sistema, asuma que la organización tiene un único vehículo y una única ruta con T terminales de venta (procesos). El vehículo tiene N asientos (inicialmente disponibles).

Se deberá tener en cuenta que cada nodo (proceso o *peer*) del sistema deberá poder conocer a los demás. Esto se puede resolver simplemente dando a cada proceso una lista de identificadores ($IP : port$) de los demás *peers* (en un archivo de configuración o en la línea de comandos).

El sistema desarrollado deberá permitir que se interactúe con él usando `telnet` como cliente, desde el cual se ingresarán comandos (`available`, `reserve n`, `cancel n`).

3. Arquitectura de un `peer`

Cada proceso (`peer`) del sistema deberá aceptar conexiones de un cliente TCP (ej: telnet), recibir y procesar comandos. Además, en cualquier momento, un proceso puede recibir mensajes de otro `peer` (por la implementación de algún algoritmo distribuido). Por esto se aconseja definir la siguiente arquitectura:

1. El `thread` principal, que procesará los eventos (arribo de mensajes) e implementa la lógica del sistema.
2. Un `thread` que acepte conexiones de un cliente tipo telnet (TCP).
3. Un `thread` que gestione las comunicaciones de otros `peers`.

La comunicación entre estos tres threads podría hacerse mediante un sistema del tipo productor-consumidor, en el cual el thread principal es el consumidor (tomando mensajes de una cola) y los otros dos son los productores (encolando mensajes).

4. Protocolo entre *peers*

Si bien la comunicación entre un proceso y un cliente se deja librado a la implementación de cada grupo, entre los peers se desea inter-operabilidad, por lo cual es necesario definir un algoritmo distribuido (protocolo) que resuelva el problema.

El problema planteado requiere que el algoritmo a utilizar garantice las invariantes del sistema. Para garantizar las invariantes, al ser un escenario concurrente, se deben atomizar las operaciones.

Para esto se deberá implementar el algoritmo de exclusión mutua distribuido de Lamport visto en el curso, con una pequeña modificación para que el mismo algoritmo permita recolectar el estado global del sistema en cada proceso.

La modificación consiste en adicionar al mensaje *RELEASE* el estado (número de asientos disponibles). De esa manera, cada proceso recibirá un mensaje con el nuevo estado del sistema en cada actualización.

A continuación se describe el protocolo a utilizar entre *peers*.

- El protocolo debe ser UDP. Asumiremos que la comunicación es confiable (no hay pérdidas de paquetes).
- Formato de mensajes (texto):
  - `ENTER ts`
  - `REPLY ts`
  - `RELEASE n ts`

donde `ts` y `n` (enteros positivos) son el *timestamp* y el número de asientos disponibles, respectivamente.

5. Evaluación

Cada grupo deberá incluir su implementación en una prueba de inter-operabilidad con las demás implementaciones y cada uno de sus integrantes deberán poder responder satisfactoriamente a las preguntas realizadas por el

equipo docente y los demás alumnos.

Los elementos de evaluación serán:

- Análisis del algoritmo propuesto (¿garantiza los invariantes?)
- Correctitud de la implementación.
- Inter-operabilidad.
- Facilidad de uso.
- Claridad de la implementación.
- Abstracciones utilizadas y definidas.
- Originalidad (cuidado con las copias).
- La ayuda requerida por parte del equipo docente.