Proyecto de Telecomunicaciones y Sistemas Distribuidos 2015

Integrantes

Luciano Putruele, Condición: Regular.
Agustin Bauer, Condición: Regular.
Alan Gonzalez, Condición: Regular.

Lenguaje utilizado

Java

El problema

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

La venta de pasajes de ómnibus (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 comunicacion (requiriendo un ancho de banda superior para el servidor) y ofrece un único punto de falla.

Sobre el programa

El algoritmo distribuido utilizado -para garantizar exclusion mutua entre los procesos- es el de los timestamps de Lamport.

El programa utiliza dos threads, uno para el servidor TCP (comunicación con el cliente) y otro para el servidor UDP (comunicación con los peers).

Por lo tanto también hacia falta sincronizar todos los accesos a la estructura utilizada por el algoritmo de Lamport, ya sean lecturas o escrituras, ya que de lo contrario se podrian dar condiciones de carrera.

El sistema está compuesto por las siguentes clases:

Main: Clase principal, inicia los threads correspondientes a UDPServer y TCPServer.
TCPServer: Maneja la comunicación con el cliente.
UDPServer: Maneja la comunicación con los peers.
Terminal: Representa el estado del sistema (cantidad de asientos disponibles), contiene los métodos para consultar,reservar y cancelar asientos.
Message: Estructura que representa una consulta, tiene un timestamp.
SyncQueue: Estructura utilizada para el algoritmo de Lamport, es una cola de prioridades sincronizada.
PeerData: Estructura que representa la información util de un peer (p.e ip).

Analisis del algoritmo de Lamport

Este algoritmo crea 3(N − 1) mensajes por consulta:

(N − 1) requests
(N − 1) replies
(N − 1) releases

Por cada 2 eventos diferentes a y b ocurriendo en el mismo proceso, es necesario que el timestamp de a nunca sea igual al de b. Por lo tanto se necesita que el reloj lógico debe ser implementado de tal manera que haya al menos un "tick" entre los eventos a y b. Cuando tenemos varios procesos, también es necesario agregar información extra, el ID del proceso, para poder diferenciar entre eventos a y b que pueden ocurrir "al mismo tiempo".

Propiedades:

Safety (nunca pasa algo malo):

Los timestamps de los requests para entrar a una zona critica de 2 procesos cualesquiera pueden ser totalmente ordenados sin ambiguedad. Entonces, cuando un proceso Pi obtiene replies de todos los demas procesos, y su request tiene el menor timestamp entre todos los requests entonces, el request de Pi va a estar en la cabeza de la cola en todos los procesos. Hasta que Pi no mande un mensaje release, ningun otro proceso puede desencolar el request de Pi, y por lo tanto no puede entrar a la zona critica.

Liveness (algo bueno pasa eventualmente) y Fairness:

El multicast de un request de un proceso Pi, se va a encolar sin problemas en todos los procesos. Cualquier request generado en los demas procesos despues de encolar el request de Pi esta asegurado que se va a encolar mas atras que el de Pi.(Fairness: Los requests se ejecutan en el orden en el que fueron generados).

Como el tiempo de ejecución de una zona crítica es finito, la cola de requests seguro se va a actualizar en todos los procesos, el request de cualquier proceso eventualmente va a estar en la cabeza de la cola.

Instrucciones de uso

1.Modificar el archivo config.txt donde cada linea debe tener el siguiente formato,
la primer linea es la informacion local: pid-udpport-tcpport, desde la segunda linea esta la información de los demas peers: ip-pid-udpport-tcpport
2.Ejecutar el programa (java -jar Main).
3.Ejecutar telnet (telnet localhost port).
4.Las consultas son available, reserve n y cancel n donde n es la cantidad de asientos a reservar o cancelar.