Sistemas Distribuidos
Tarea 1

Universidad Técnica Federico Santa María
Departamento de Informática

José García
<jose.garcia.14@sansano.usm.cl>

Juan León
<juan.leonl.14@sansano.usm.cl>

Paulina Silva
<pasilva@inf.utfsm.cl>

2 de octubre de 2019

1. Introducción: Simulando un ChromeCast

En la siguiente situación se tiene un dispositivo ChromeCast que se usa para reproducir contenido multimedia, el cual posee una conexión a una red y se comunica con los clientes interesados mediante el uso de *Broadcast*. Muchos consideran este tipo de comunicación poco adecuada, debido a que cualquier cliente conectado a la red podría dar comandos al ChromeCast (Siempre y cuando pueda hacerlo). Para esta tarea, se simulará la situación usando **java** y la clase **java.net.MulticastSocket**, reemplazando el funcionamiento de *Broadcast* por un grupo *Multicast*.

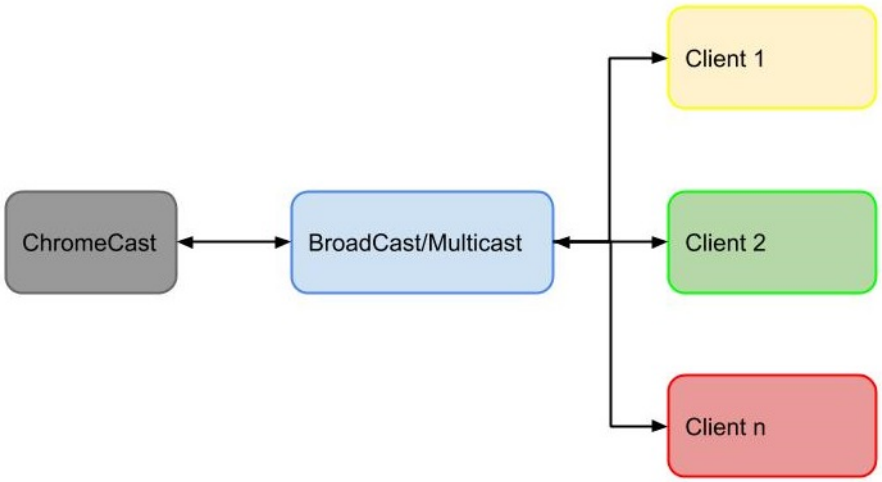

Figura 1: Esquema de comunicación Multicast para esta tarea

Los procesos de cliente y servidor serán ejecutados en máquinas virtuales, las que serán asignadas a cada grupo por los ayudantes.

2. Desarrollo

Para el desarrollo de esta tarea deberá usar **Java 8**. Recuerde revisar la versión por defecto del compilador usado.

2.1. Implementar un servidor (ChromeCast)

Este servidor debe cumplir con:

1. Ejecutar las instrucciones que un cliente le solicite. Estas serán:
   - **Play**: En esta acción, el servidor deberá reproducir el contenido que un cliente solicite. Para esta tarea, bastará con que repita el contenido del mensaje cada 1 segundo, y señale el progreso de la reproducción en base a la duración del contenido.
   - **Stop**: Detiene la ejecución actual, y devuelve al servidor a su estado inicial (cola vacía).
   - **Pause**: Detiene la acción de reproducción actual. Si es llamado por segunda vez, reanuda la reproducción.
   - **Queue**: En esta acción, en vez de reproducir el contenido, lo almacena en un cola, en la cual los contenidos serán reproducidos a medida que los anteriores terminen de reproducirse.
   - **Next**: Avanza al siguiente contenido en la cola.
   - **Jump**: Salta a un contenido específico de la cola. En este caso, el argumento de esta instrucción es la posición en la cola, y el servidor pasa a reproducir ese contenido. Todos los contenidos posicionados antes de este se desechan.
2. Comunicar su estado al grupo multicast bajo la siguientes normas:
   - **Play**: Mencionado anteriormente, en este estado el ChromeCast debe repetir el contenido de un mensaje cada 1 segundo, agregando al mensaje el progreso de la reproducción.
   - **Pause**: El ChromeCast debe comunicar al grupo que se encuentra pausado, y también indicar como reanudar la reproducción.
   - **Stop**: El ChromeCast indica al grupo que se ha detenido.
3. Particularidades
   - Como ejemplo, se sugiere usar la siguiente estructura para el servidor:
     user@user:~$ java server <ip multicast>
     En caso contrario, especificar en un archivo README como iniciar el servidor.
   - Respecto del mensaje emitido por el ChromeCast, este debe incluir el emisor, la instrucción (solo Play, Pause y Stop se muestran, las demás se ejecutan sin ser mostradas), el **contenido** (mensaje) si corresponde, el progreso del contenido, y un ID que permita ordenar los mensajes (Usar algún delimitador para poder procesar el mensaje posteriormente). Como ejemplos:
     >CCast_Play_*Enter Sandman - Metallica*_20 %_ID:000256
     >CCast_Pause_(Pausa para reanudar)_ID:000257
     >CCast_Stop_ID:000258

2.2. Implementar un cliente

Los clientes que se quieran unir al grupo deben cumplir con:

- Poder unirse al grupo Multicast y ser capaz de recibir los mensajes que el servidor envía a la dirección del grupo.
- Almacenar el historial de comandos y la cola de reproducción, indicando cliente y comando ejecutado, y cliente y contenido, respectivamente. Para esto, el cliente debe ser capaz de almacenar las instrucciones propias, y las instrucciones de otros clientes.
- Ver la cola por linea de comandos. Esto implica que se debe crear una instrucción específica para los clientes. La instrucción será **Queue**, sin argumentos, es decir, sin argumentos se consulta la cola, y con argumentos se agrega contenido a la cola.
- Ver el historial por linea de comandos. Esto implica que se debe crear una instrucción específica para los clientes, llamada **History**, la cual muestre el historial en la linea de comandos.
- Como ejemplo, se sugiere usar la siguiente estructura para el Cliente:
  user@user:~$ java client <ip multicast>
  En caso contrario, especificar en un archivo README como iniciar el cliente
- Respecto del mensaje emitido por el cliente, este debe incluir el emisor, la instrucción, el contenido (mensaje) y la duración (en segundos) si corresponde y un ID que permita ordenar los mensajes (Usar algún delimitador para poder procesar el mensaje posteriormente). Como ejemplos:
  >Client1_Play_*Sign - FLOW*_120_ID:000004
  >Client1_Queue_*Numb - Linkin Park*_160_ID:000005
  >Client1_Pause_ID:000006
  >Client1_Next_ID:000007
  >Client1_Stop_ID:000008
  >Client1_History_ID:000009

3. Consideraciones

- Sin importar la cantidad de instanciaciones del cliente, el servidor debe poder enviar un mensaje al grupo y todos los clientes unidos a éste deben poder recibir el mensaje.
- Recuerden tener en cuenta que mientras se recibe un paquete por un thread (para los mensajes del grupo multicast), es posible usar otra thread para solicitar las distintas instrucciones, de manera que aunque el thread que escucha en un socket se bloquee (por que espera a que lleguen mensajes multicast), otro thread podrá hacerse cargo de las instrucciones respectivas.
- Recomendación: Si utiliza varios threads para cada una de las tareas del cliente y recibir paquetes del grupo multicast, se recomienda ocupar *Handlers* para pasar información de un thread a otra.
- La entrega de la tarea debe incluir un archivo Makefile, el cual debe contar con el target default que compile las clases necesarias para ejecutar servidor y cliente, y con el target clean que borre todos los archivos compilados.
- También debe incluir un archivo README, en el cual debe especificar:
  - Integrantes y ROL USM.
  - Puerto y dirección IP usados durante el desarrollo para la conexión entre procesos.
  - Especificaciones en caso de que los argumentos de cada clase tengan una estructura distinta a la sugerida.
  - Instrucciones de ejecución de los scripts.

4. Condiciones

- La tarea debe ser realizada en grupos de 2 personas.
- La versión de Java a utilizar debe ser Java 8. Errores por la versión de Java no serán corregidos.
- Fecha de entrega: **De acuerdo a votación.**
- En caso que se descubra copia, **esto equivale a nota 0 para los estudiantes implicados**.
- Por cada día de atraso se descuentan 20 puntos, hasta un máximo de dos días de retraso, posterior a ese plazo no se moleste en entregarla pues la nota final será 0.
- La tarea debe ser subida en la fecha correspondiente a las máquinas virtuales. Deben ser subidos todos los archivos en las máquinas virtuales, de manera que cualquiera pueda cumplir el rol de servidor o cliente.
- Para cualquier duda, inquietud o reclamo, se ha creado un foro en la sección *Entregas*. No dude en hacer llegar sus consultas, o revisar las consultas anteriores en caso de dudas. Link del foro.