Sistemas Distribuidos
Tarea 2

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>

13 de octubre de 2019

1. Introducción: Extracción Distribuida

Se deben crear procesos que participen en un sistema de extracción de recursos basado en el Algoritmo de Suzuki-Kasami. Se crearan N procesos que formarán parte de este algoritmo. El programa mostrará como salida un color que representa el estado del lote de recursos. El lote de recursos puede tener los siguientes estados, así como lo muestra la Figura 1:

- Rojo: Queda menos del 25 % de los recursos iniciales.
- Amarillo: Queda entre 50 % y 25 % de los recursos iniciales.
- Verde: Queda entre 75 % y 50 % de los recursos iniciales.
- Azul: Queda entre 100 % y 75 % de los recursos iniciales.

El sistema de monitoreo de los recursos necesita implementar un solo Token, es decir, existe una sola instancia para todo el sistema que puede extraer recursos a la vez, por lo que usted debe hacer uso de RMI para lograr este objetivo. Un ejemplo de la situación se puede ver en el siguiente dibujo:

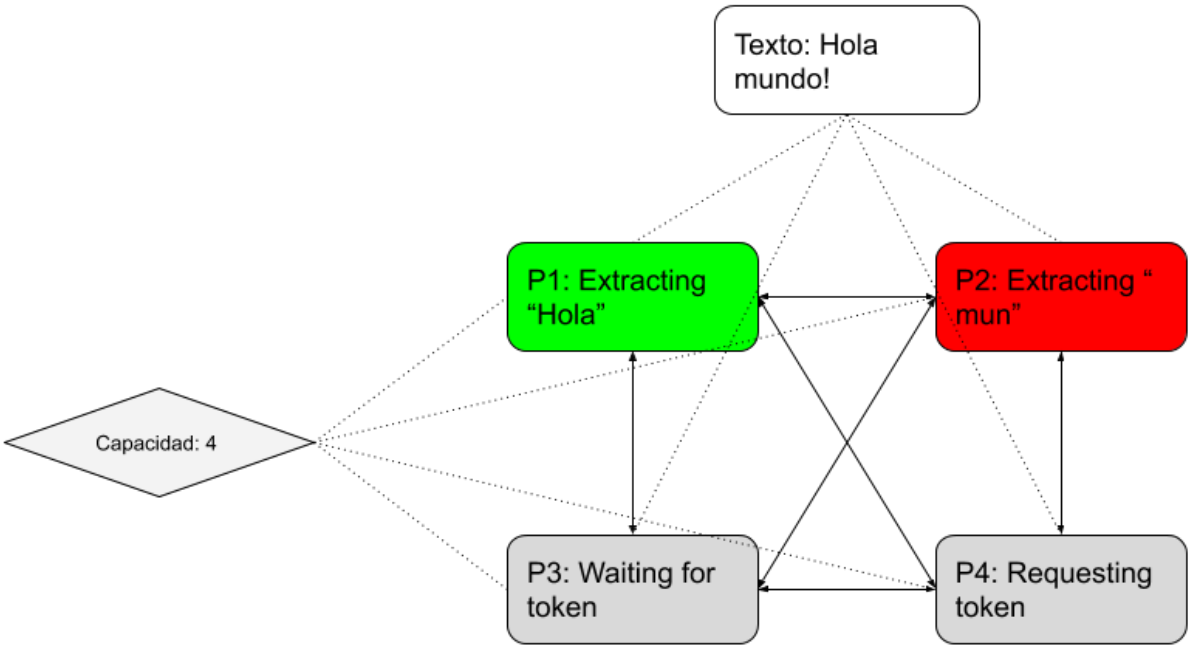

Figura 1: Caso de ejemplo con 1 sola zona crítica. El proceso 1 ya fue ejecutado, el proceso 2 extrae las letras y tiene el token, el proceso 3 está a la espera del token, y el proceso 4 aún espera el token.

2. Desarrollo

Para el desarrollo de esta tarea deberá usar **Java 8**. Recuerde revisar la versión por defecto del compilador usado. Un mismo proceso será ejecutado N veces en una misma máquina, y estos N procesos deben comunicarse por medio de RMI.

2.1. Recursos

El proceso recibirá un archivo de texto, el cual representará los recursos a extraer. Los recursos a extraer serán simulados por las letras del texto.

2.2. Proceso extractor

El proceso extractor debe cumplir con lo siguiente:

1. Extraer letras. El proceso debe cumplir con abrir el archivo de texto y extraer la primer letra (almacenarla en un string y borrarla del texto). El almacenamiento no se especifica, pero el proceso debe ser capaz de mostrar las letras extraídas en cada sección crítica (de forma distinguible) al final de la ejecución.
2. Capacidad y enfriamiento. El proceso solo podrá extraer una cierta cantidad de letras en una sección crítica. Luego de realizar la extracción, el proceso tiene que esperar un tiempo de *enfriamiento* antes de volver a pedir el token. Este tiempo de enfriamiento es fijo, y es la mitad de la capacidad en segundos.
3. Velocidad. El proceso debe extraer las letras, 1 a la vez, según se especifique en la velocidad (letras/-segundo).
4. Ejecución. El proceso debe ser ejecutado de la siguiente forma:
   user@user:~$ java process N <archivo><capacidad><velocidad><delay><bearer>.
   A continuación se explica cada argumento:
   - **N**: Cantidad de procesos a ser ejecutados.
   - **archivo**: Este argumento es el nombre del texto a procesar.
   - **capacidad**: Número natural que representa la cantidad de letras que puede extraer cada proceso en cada sección crítica.
   - **velocidad**: Cantidad de letras que puede procesar el programa. Como el tiempo de procesado de archivos es muy bajo, este valor será aplicado en un *sleep()* para simular la velocidad real. Por ejemplo, si la velocidad es 3, el tiempo para la función *sleep()* será 0,33 segundos o 333 milisegundos (asumiendo que el tiempo de procesamiento del archivo es despreciable).
   - **delay**: Valor en milisegundos que indica la demora en inicio del proceso (para poder sincronizar ejecución de procesos).
   - **bearer**: Boolean (True o False) que indica si el proceso ejecutado posee el Token inicialmente. Solo un proceso tendrá el Token en la ejecución inicial.

2.3. RMI

Para poder implementar el algoritmo usando RMI, es necesario que los procesos implementen los siguientes métodos:

- request(id, seq): Método que registra un request de un proceso remoto. Recibe el id del proceso que hace la petición y el número de petición del proceso.
- waitToken(): Método que le indica a un proceso remoto que debe esperar por el token para realizar la sección crítica.
- takeToken(token): Método que toma posesión del token en el proceso.
- kill(): Método que mata el proceso remoto. Debe usar este método para detener el algoritmo de S-K una vez que el token haya pasado por todos los nodos del sistema.

P.D: Si desea implementar más métodos que ayuden en su trabajo, usted es libre de hacerlo en la medida que explique su utilidad detalladamente en el archivo README.

2.4. Output

El proceso debe informar su estado por consola. Los estados son:

- Ocioso: Todo proceso parte en este estado. Se considera a un proceso ocioso cuando no está ejecutando la sección crítica, ni esperando Token. Informar este estado mediante un texto en la consola.
- Esperando Token: Este estado ocurre cuando el método waitToken() es invocado, y el proceso se prepara para entrar a la sección crítica. Informar este estado mediante un texto en la consola.
- Sección crítica: En este estado, se procede a extraer las letras. Informar este estado mediante colores en la consola, especificados anteriormente.

Además del estado, el proceso debe informar por consola el arreglo RN.

3. Consideraciones

- Cada proceso ejecutará un número variable de secciones críticas, por lo que la cantidad de secciones estará determinada por el largo del texto a procesar, es decir, cuando el texto se acabe, se debe dar por terminada la ejecución.
- El Token también puede ser usado para transmitir información. Puede usarlo para comunicar la cantidad inicial de recurso entre los procesos.
- Toda operación que involucre comunicación entre procesos será realizada por medio de RMI utilizando al menos los métodos expuestos anteriormente.
- Recomendación: Si tiene dudas del funcionamiento del algoritmo y la situación a simular, investigue los conceptos *Condición de carrera* y *Exclusión mutua*.
- 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.
  - Instrucciones de ejecución de los scripts.
- La máquina a usar para esta tarea puede ser cualquiera de las 2 máquinas de la tarea anterior. Por esta razón, suba su entrega a ambas máquinas virtuales.

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 con la ejecución.
- 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.