Programación Concurrente 2020

Trabajo Práctico Final

Condiciones

- El trabajo es grupal, de 3 a 4 alumnos (5 es la excepción).
- Mientras dure la Situación de Emergencia Pública en Materia Sanitaria la defensa del trabajo es a través de videoconferencia.
- La evaluación es individual (hay una calificación particular para cada integrante)
- Solo se corrigen los trabajos que hayan sido subidos al aula virtual (LEV)
- Los problemas de concurrencia deben estar correctamente resueltos y explicados.
- El trabajo debe implementarse en lenguaje Java.
- Se evaluará la utilización de objetos y colecciones, como así también la explicación de los conceptos relacionados a la programación concurrente.

Red de Petri

Las transiciones con relleno negro son inmediatas, mientras que las transiciones sin relleno, son temporizadas.

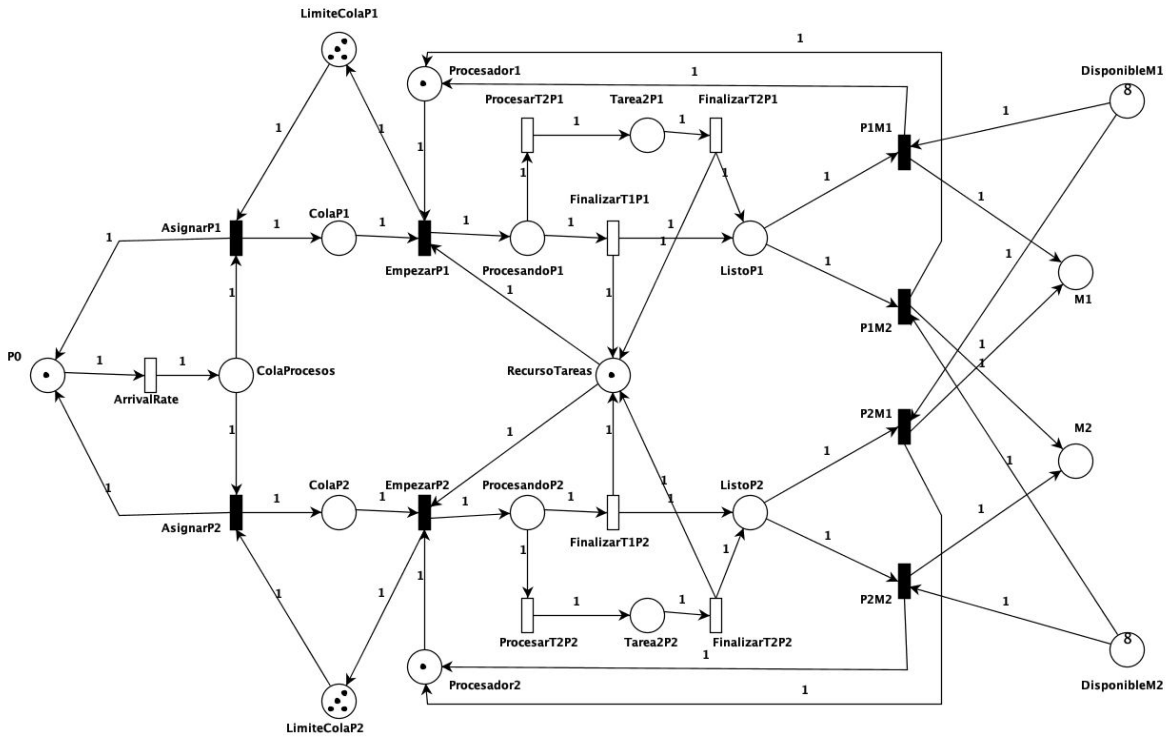

Figura 1

Enunciado

En la Figura 1 se observa una red de Petri con el modelado de un sistema con dos procesadores con sus respectivas colas de tareas asociadas y dos slots de memoria para el almacenamiento de datos ya procesados. Los principales componentes de la red de la Figura 1 son:

- **ColaProcesos**, la cual almacena las tareas que arriban a procesarse. Esta plaza habilita dos transiciones que representan la asignación de la tarea para ser procesada por un determinado procesador.
- **ColaPx**, dos plazas que representan las colas de tareas asociadas a cada procesador. También poseen una plaza asociada **LimiteColaPx**, la cual modela la capacidad máxima de cada cola.
- **Procesadorx**, plazas que modelan los recursos procesadoradores.
- Para procesar las tareas es necesario el recurso **Procesadorx** más un recurso compartido único intrínseco del sistema, el cual se representa con la plaza **RecursoTareas**.
- **ProcesandoPx**, plazas que representan la situación de realización de tareas del tipo 1 o del tipo 2. Ver detalle de Tareas.
- Luego de procesar las tareas, se marcan las plazas **ListoPx**, las cuales habilitan las transiciones relacionadas a cada slot de memoria.
- Las plazas **Mx** representan los slots de memoria, y las plazas **DisponibleMx** la capacidad máxima asociada a cada slot de memoria.

Tareas

Las tareas que arriban para procesarse pertenecen a uno de los siguiente tipos:

- Tareas tipo 1: tareas que requieren de una sola subtarea temporal. Completar una tarea de este tipo está representado en la Figura 1 con la ejecución de las transiciones **FinalizarT1Px**.
- Tareas tipo 2: tareas que requieren de dos subtareas temporales. Completar una tarea de este tipo está representado en la Figura 1 con la ejecución de las transiciones **ProcesarT2Px** y **FinalizarT2Px**.

La suma de los tiempos asignados a las transiciones relacionadas a las tareas de tipo T2, debe ser mayor al tiempo asignado a la transición de tipo T1.
Es decir:

- Tiempo de **ProcesarT2Px** + tiempo de **FinalizarT2Px** > tiempo de **FinalizarT1Px**

Memorias

Las memorias deben modelarse con los siguientes lineamientos:

- Capacidad máxima de 8 datos en cada una
- Su vaciado debe realizarse en un tiempo entre *alfa* y *beta* milisegundos, a definir según criterio del grupo.

Políticas

Es necesario para el modelado del sistema implementar políticas que resuelvan los conflictos relacionados a:

- Mantener la carga en los procesadores balanceada.
- Mantener la carga en las memorias balanceada.

Requerimientos

1) Implementar la red de Petri de la Figura 1 haciendo uso de una herramienta, ej: **PIPE**. Verificar todas sus propiedades, hacer un análisis y en caso que exista deadlock en la red provista, implementar los cambios necesarios para liberar la red del deadlock.
2) El proyecto debe ser modelado con objetos en Java, haciendo uso de un monitor de concurrencia para guiar la ejecución de la red de Petri.
3) Implementar un objeto Política que cumpla con los objetivos establecidos en el apartado Políticas.
4) Hacer el diagrama de clases que modele el sistema.
5) Hacer el diagrama de secuencia que muestre el disparo exitoso de una transición que esté sensibilizada, mostrando el uso de la política.
6) Indicar la cantidad de hilos necesarios para la ejecución y justificar.
7) Realizar múltiples ejecuciones con 1000 tareas completadas (para cada ejecución), y demostrar con los resultados obtenidos:
    a) Cuán equitativa es la política implementada en el balance de carga en los procesadores.
    b) Cuán equitativa es la política implementada en el balance de carga en las memorias.
    c) La cantidad de tareas de cada tipo que ejecutó cada procesador, justificando el resultado.
    d) La cantidad de datos que almacenó cada slot de memoria.
8) Registrar los resultados del punto **7)** haciendo uso de un archivo de log para su posterior análisis.
9) Mostrar e interpretar los invariantes de plazas y transiciones que posee la red.
10) Verificar el cumplimiento de los invariantes de plazas luego de cada disparo de la red.
11) Verificar el cumplimiento de los invariantes de transiciones mediante el análisis de un archivo log de las transiciones disparadas al finalizar la ejecución. Sugerencia: hacer uso de expresiones regulares.
12) El programa debe poseer una clase Main que al correrla, inicie el programa.

Entregables

a) Un archivo de imagen con el diagrama de clases, en buena calidad.
b) Un archivo de imagen con el diagrama de secuencias, en buena calidad.
c) En caso de que el punto 1) de los requerimientos requiera modificar la red de Petri, se debe entregar:
    i) Un archivo de imagen con la red de Petri final, en buena calidad.
    ii) El archivo fuente de la herramienta con la que se modeló la red.
d) El código fuente Java (proyecto) de la resolución del ejercicio.
e) Un informe obligatorio que documente lo realizado, los criterios adoptados y que explique los resultados obtenidos.

Subir al LEV el trabajo **TODOS** los participantes del grupo.

Fecha de entrega

16 de Junio de 2020