FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES
UNIVERSIDAD NACIONAL DE CÓRDOBA

Programación Concurrente
Trabajo Práctico III

Redes de Petri para simular un procesador de 2 núcleos

Docentes:

- Dr. Ing. Micolini Orlando
- Ing. Ventre, Luis
- Ing. Mauricio, Ludemann

Benitez, Darío Jeremías
Perez, Bruno Santiago

20 de julio de 2020

Índice

1. Consigna

Se debe implementar un simulador de un procesador con dos núcleos. A partir de la red de Petri de la figura 1, la cual representa a un procesador mono núcleo, se deberá extender la misma a una red que modele un procesador con dos núcleos. Además, se debe implementar una Política que resuelva los conflictos que se generan con las transiciones que alimentan los buffers de los núcleos (CPU_Buffer).

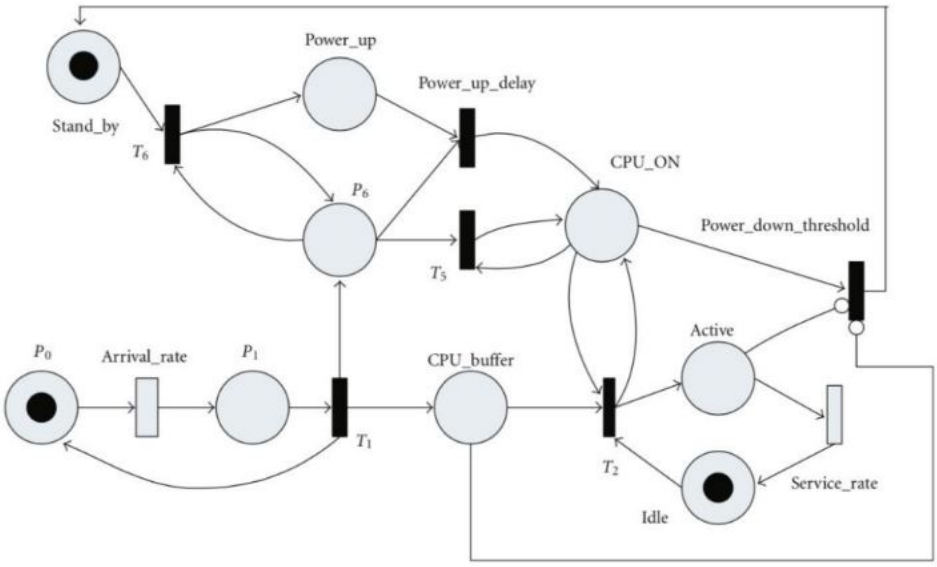

Figura 1: Red de petri para un procesador mononúcleo

Nota: Las transiciones “Arrival_rate” y “Service_rate” son temporizadas y representan el tiempo de arribo de una tarea y el tiempo de servicio para concluir una tarea.

2. Red de petri resultante

Para lograr un mejor entendimiento del problema se pasó la Red de Petri planteada en la consigna al software PIPE, se extendió la red para que consista de 2 procesadores y se renombraron las transiciones y plazas para poder seguir mejor el estado de la red.

La red resultante fue la siguiente:

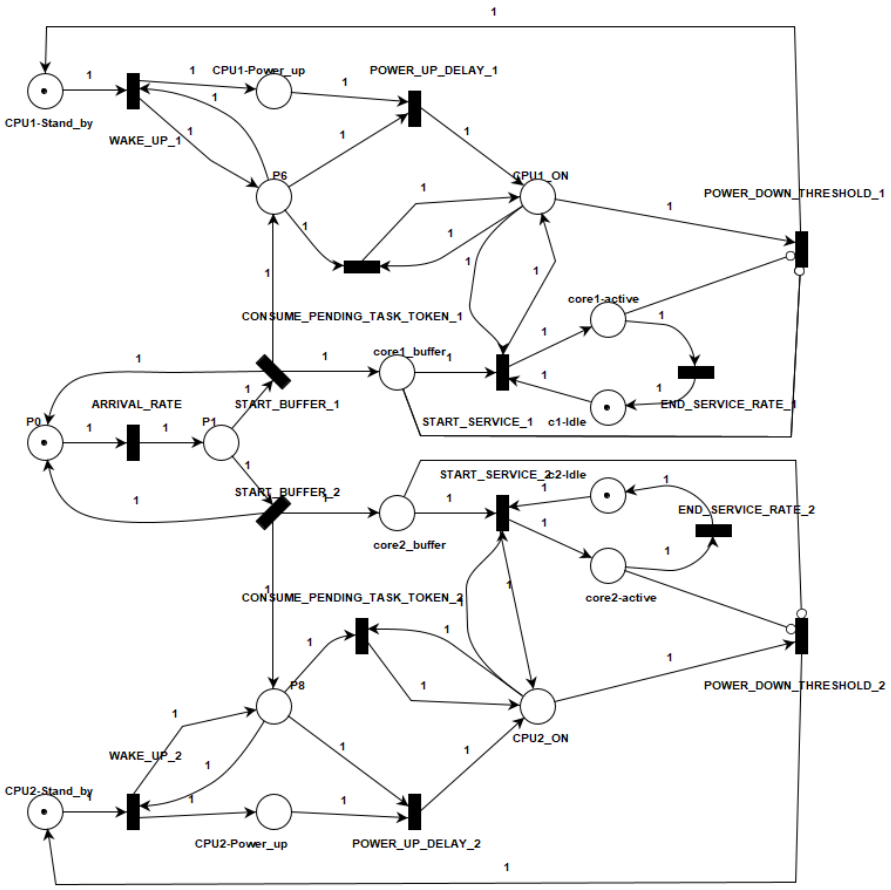

Figura 2: Red de petri extendida de 2 núcleos

3. Diseño

3.1. Diagrama de clases

Para poder llevar a cabo la implementación de la red en JAVA se realizó un diagrama de clases para tener en claro las relaciones entre los objetos.

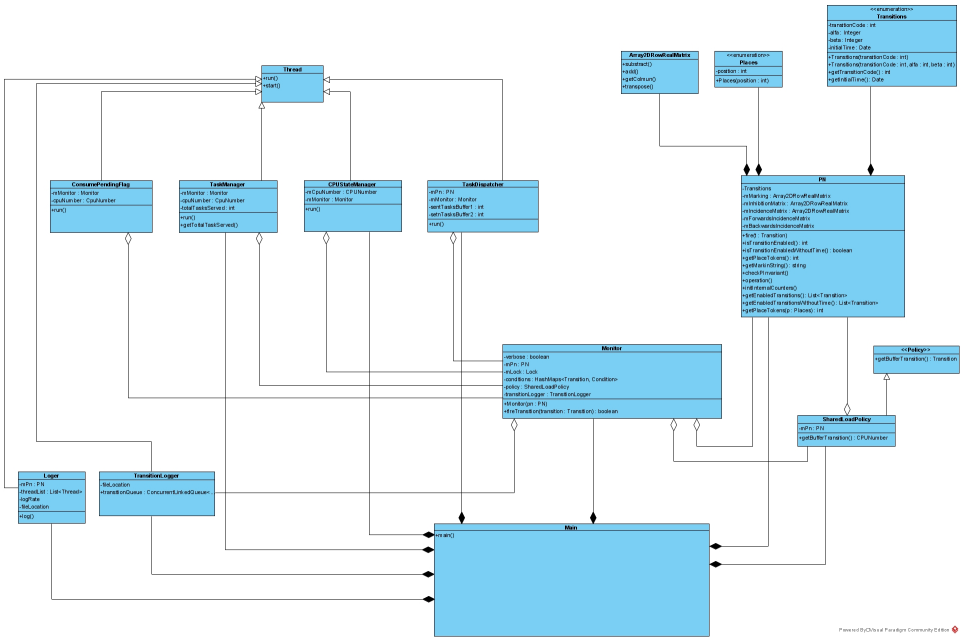

Figura 3: Diagrama de clases del proyecto. Se puede ver en los adjuntos como *diagrama_clases.jpg*.

3.2. PN (red de petri)

Esta clase contiene la funcionalidad la red de Petri. Tiene 2 elementos claves

Tiene 3 clases fundamentales

1. 4 Matrices:
   - mIncidenceMatrix
   - mForwardsIncidenceMatrix
   - mBackwardsIncidenceMatrix
   - mInhibitionMatrix
   - mMarking
2. Places: contiene las plazas de la RdP
3. Transitions: enumeración que tiene información sobre las trancisiones, su relacion respecto a las matrices y su alfa y beta

3.3. Monitor

Es el mecanismo que nos permite acceder a la red de petri de manera concurrente y efectuar disparos.

Tiene una referencia a la clase Red de Petri para disparar transiciones mediante el método fire. Además tiene un método taskdispatch que despacha las tareas al CPU correspondiente de acuerdo a la política. Esto implica un acomplamiento indeseado

entre el monitor y el problema en sí, pero como la política debía leer el estado de la red tuvimos que poner esa lógica en el monitor.

3.4. Hilos que manejan la RdP

Además hay 4 clases que conformarán 7 hilos:

1. ConsumePendingFlag (2 hilos para las 2 CPU)
2. TaskManager (2 hilos para las 2 CPU)
3. CPUStateManager (2 hilos para las 2 CPU)
4. TaskDispatcher (1 solo hilo)

3.5. Loggers

Para el desarrollo del proyecto se necesitó de 2 loggers:

1. Logger: es el encargado de cada cierta frecuencia loguear el estado de los hilos, el marcado, y la carga en los buffers
2. TransitionLogger: es el encargado de registar las transiciones disparadas y guardarlo en un archivo. Hace uso de la clase ConcurrentLinkedQueue<String> para guardar las transiciones a medida que son disparadas y consumir esa cola a medida que se puede ir imprimiendo las transiciones en un archivo.

3.6. Policy

Se define una interfaz Policy que define un contrato en el cual se debe especificar qué transicion de buffer disparar. Tenemos una sola implementación que es la clase SharedLoadPolicy. Como el nombre lo indica actúa como un balanceador de carga, es decir que va a intentar elegir la CPU que esté menos congestionada y en caso de que estén igualmente congestionadas elige la CPU más veloz.

4. Implementación

4.1. Main

Contiene las referencias los hilos que disparan transiciones a través del monitor, al monitor en sí, a la RdP y a los Loggers. Se encarga de inicializar el sistema y de finalizarlo.

4.1.1. Hilos disparadores de transiciones

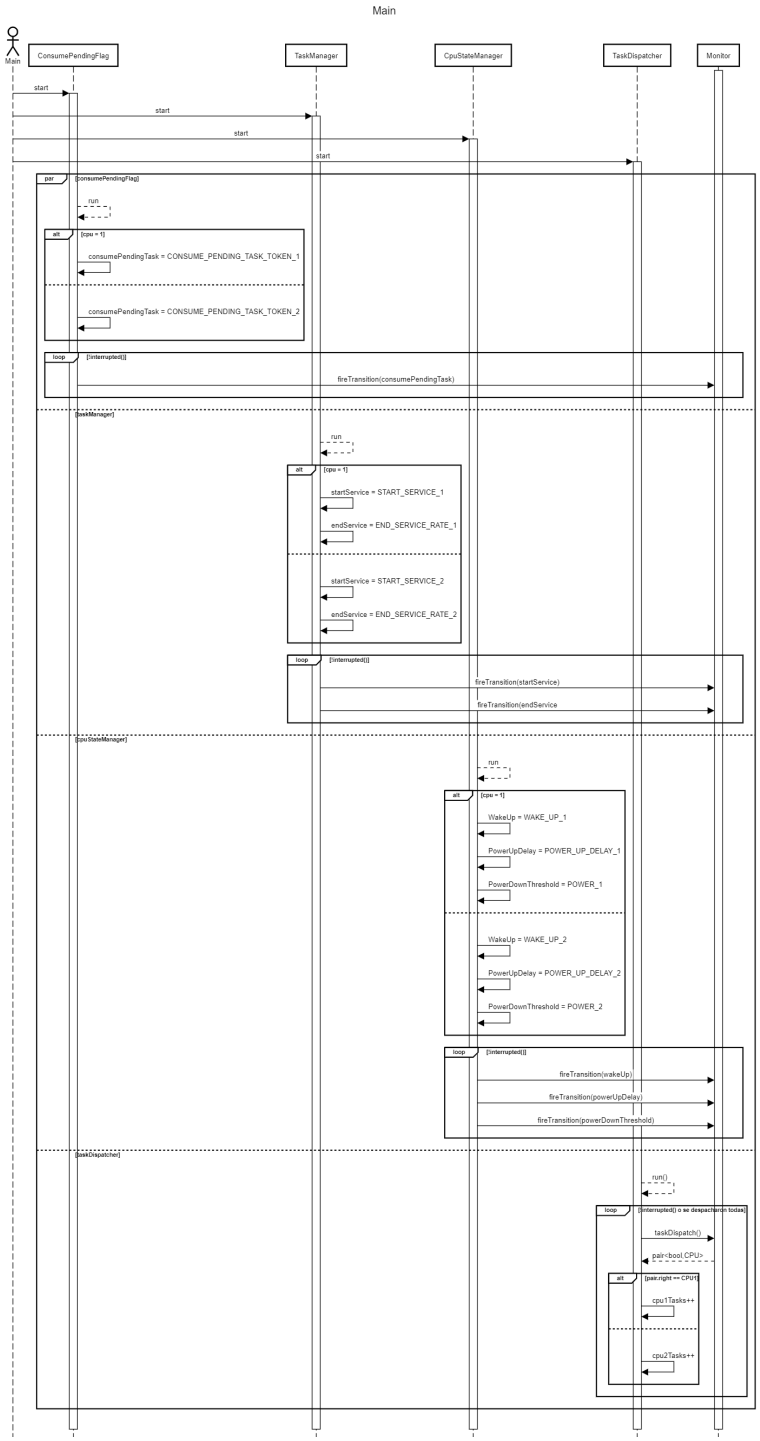

Figura 4: Diagrama de secuencia Main enfocandose en los disparos efectuados a traves del monitor

4.2. PN

Para la implementación se comenzó con la red de Petri. Se explicarán los métodos más importantes.

4.2.1. IsTransitionEnabled(t)

Se encarga de decir si una transición t está habilitada sea o no temporizada. Respuestas:

- 1 si está habilitada sea o no temporizada
- 0 si no está habilitada o se pasó del tiempo beta
- -{tiempo} si no está habilitada pero en {tiempo} milisegundos se habilitará

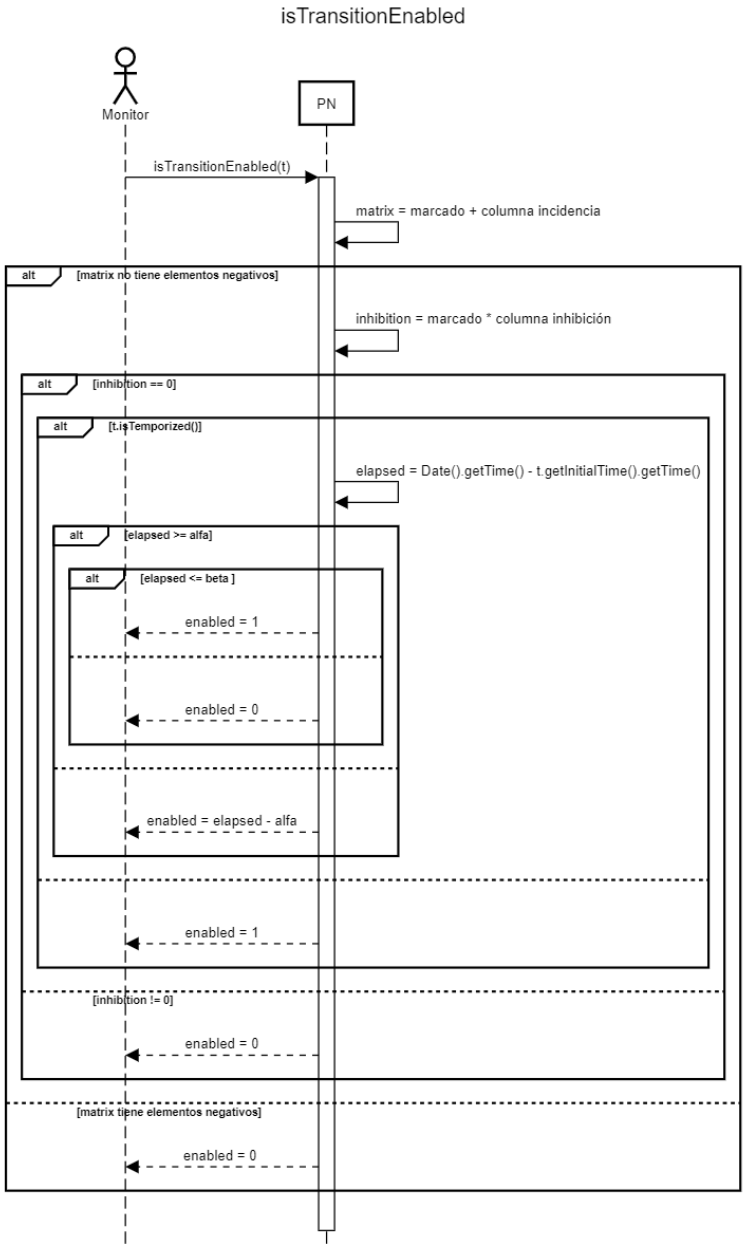

Figura 5: Diagrama de secuencia de IsTranstionEnabled

4.3. Monitor

4.3.1. Fire

Se encarga de disparar una transición. Devuelve True si se pudo ejectutar el disparo, False si no se pudo.

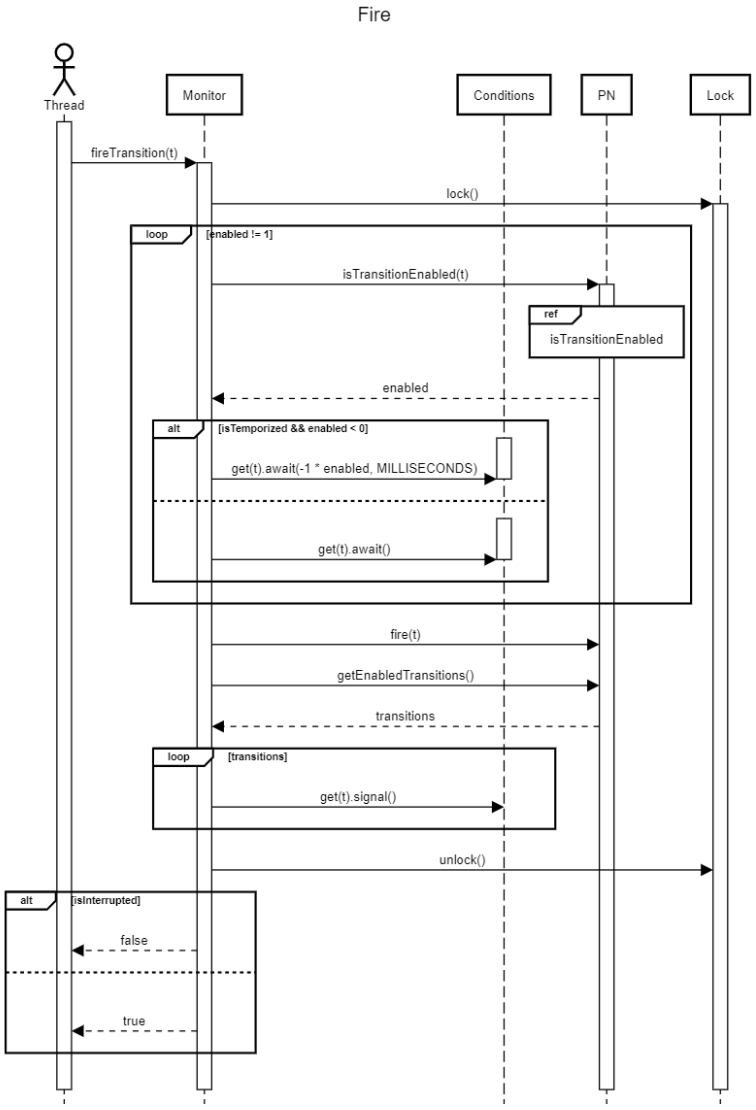

Figura 6: Diagrama de secuencia de fire

4.3.2. TaskDispatch

Se utiliza para despachar una tarea. Utiliza al objeto política para determinar hacia qué buffer despachar una tarea.

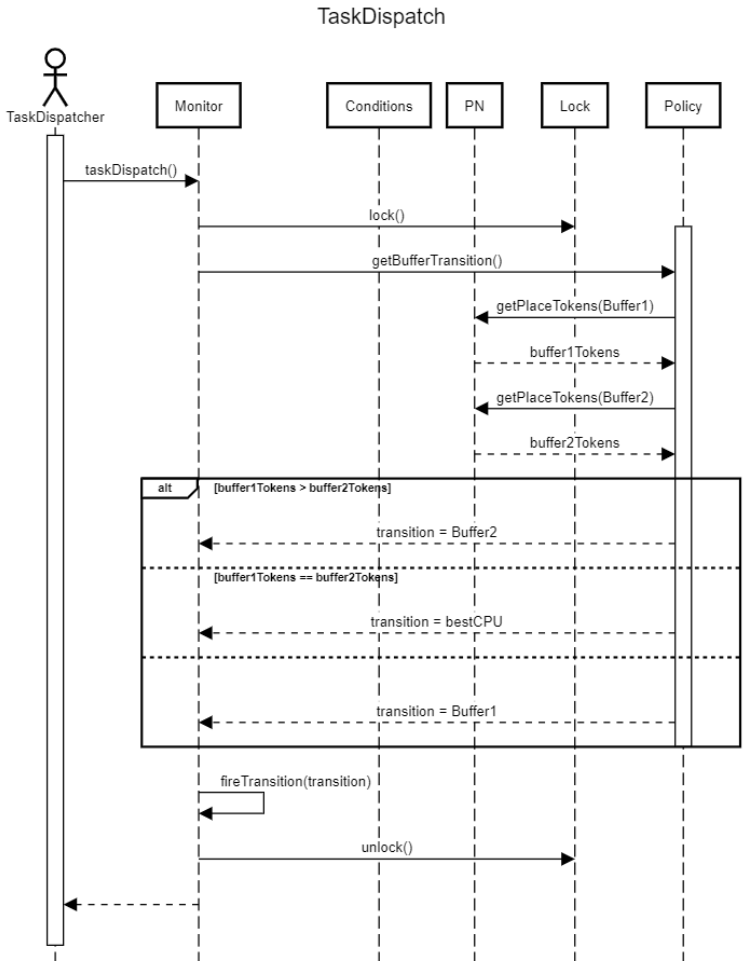

Figura 7: Diagrama de secuencia de taskdispatch