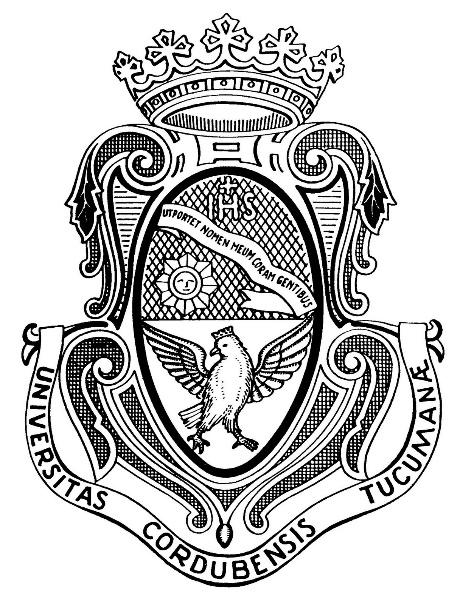
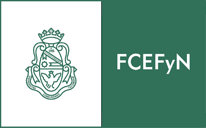
UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES





Trabajo Práctico Nº 3

Materia: Programación Concurrente

Grupo: ThreadRipper

**Año 2019**

|  |  |
| --- | --- |
| **Alumnos** | **Matrícula** |
| **García,** Bruno Emilio | 39449179 |
| **Oroná,** Madelein Ayelen | 37852092 |
| **Losano Quintana,** Juan Cruz | 39621464 |
| **Vignolo**, Gabriel Enrique | 39080905 |

1. **Enunciado:**

Se implementará un simulador de un procesador de dos núcleos en el cual, a partir de la red de Petri de la figura 1 propuesta para un procesador mono núcleo, se extenderá para el caso ya mencionado.

Además, se usarán políticas para resolver los conflictos que se generan con las transiciones que alimentarán a los buffers de los núcleos.

Las transiciones “Arrival\_Rate” y “Service\_rate” son temporizadas y representan el tiempo de arribo de procesos y el tiempo de servicio que tendrá para concluir una tarea.

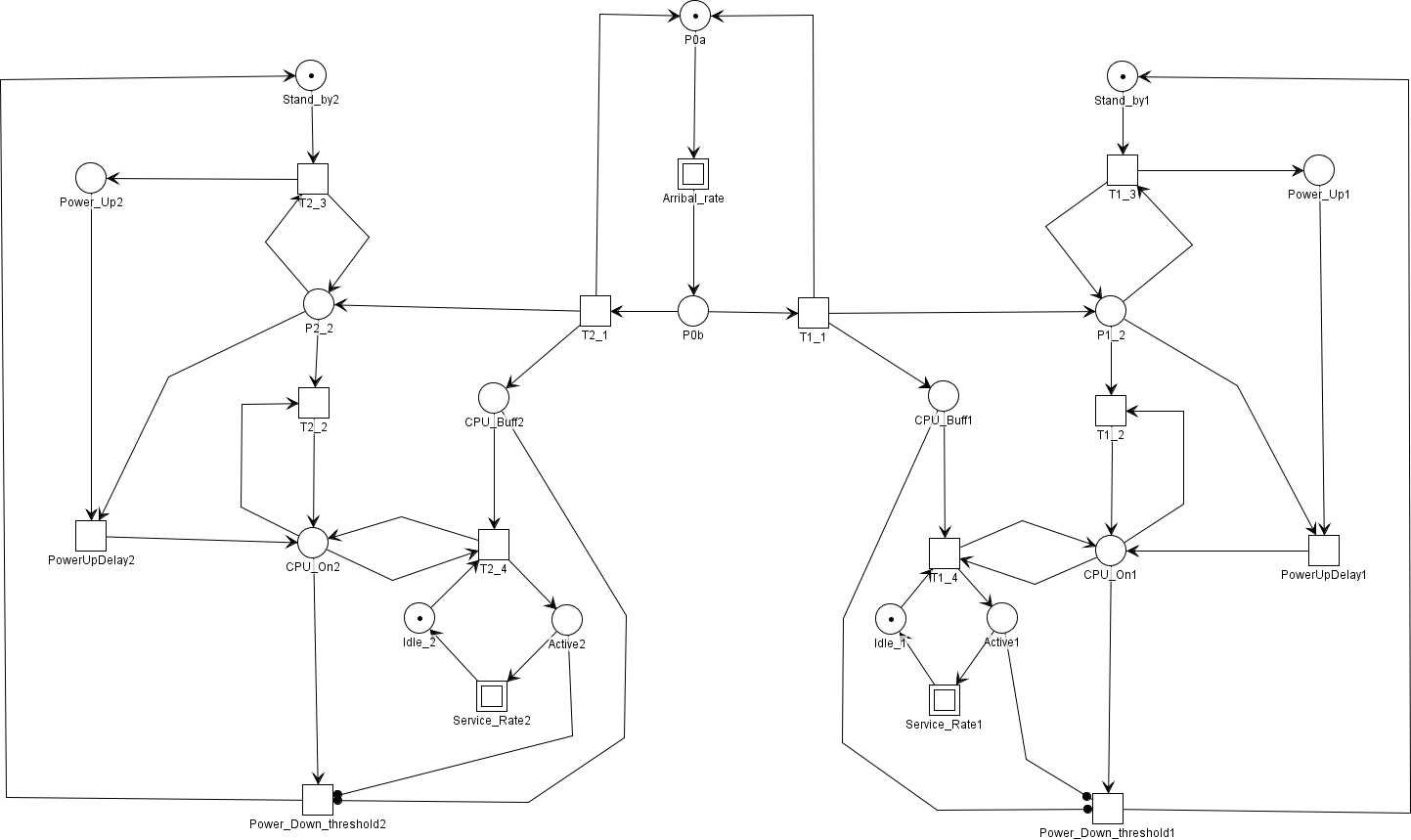


**Figura 1. Modelo de Red de Petri de una CPU mono núcleo.**

1. **Consignas:**

* Modelar el sistema de un procesador con dos núcleos, extendiendo la RdP de la ***Figura 1*** y verificar todas sus propiedades haciendo uso de la herramienta *“Petrinator”*.
* Hacer un diagrama de clases que modele el sistema.
* Implementar un objeto “Política” que resuelva el conflicto entre las transiciones sensibilizadas que alimentan los buffers de los núcleos, manteniendo la carga de la CPU equitativa.
* Hacer un diagrama de secuencia que muestre el disparo exitoso de una de las transiciones que alimenta a uno de los buffers de la CPU, mostrando el uso de la política.
* Indicar la cantidad de hilos necesarios para la ejecución y justificar.
* Modelar el sistema con objetos en Java.
* Realizar las siguientes ejecuciones con 1000 tareas completadas
  + - Ambos núcleos con el mismo tiempo de “Service\_Rate”.
    - Un núcleo con el doble de tiempo de “Service\_Rate” que el otro.
    - Un núcleo con el triple de tiempo de “Service\_Rate” que el otro.
* Verificar los resultados haciendo uso de un archivo de Log, verificando el cumplimiento de los invariantes.

1. Modelado del problema con Redes de Petri:



1. **Desarrollo:**

Para el caso analizado, realizaremos las siguientes suposiciones:

1. El arribo de tareas seguirá una distribución exponencial con media
2. El tiempo de servicio de tareas seguirá una distribución exponencial con media .
3. La CPU entra en modo de “Stand\_by” si no hay más tareas para ser atendidas después de un período T (por medio de Power Down Threshold) y si no hay procesos que estén utilizando el núcleo.

Las transiciones del modelo que se observa en la Figura 1, consiste en un mix de transiciones temporales, inmediatas y deterministas. En la Tabla 1, se presentan las distribuciones y prioridades de estas transiciones, para el caso de ambos núcleos con el mismo tiempo de “Service\_Rate”.

|  |  |  |
| --- | --- | --- |
| **Transiciones** | **Distribución de disparo** | **Prioridad** |
| Arrival\_Rate1 & Arrival\_Rate2 | Exponencial | N/A |
| T1\_1 & T2\_1 | Instantánea | 4 |
| T1\_4 & T2\_4 | Instantánea | 1 |
| Service\_Rate1 & Service\_Rate2 | Exponencial | N/A |
| PDT\_1 & PDT\_2 | Determinista | N/A |
| T1\_2 & T2\_2 | Instantánea | 2 |
| T1\_3 & T2\_3 | Instantánea | 3 |
| PUD\_1 & PUD\_2 | Determinista | N/A |

Para el caso de que se duplique o triplique el tiempo de “Service\_Rate”, optamos por darle mayor prioridad a la transición T1\_1 o T2\_1. Dependiendo de cuál de las ramas tendrá más demora, tendrá más prioridad de atención la rama opuesta, esperando así que aumente el rendimiento, o se mantenga equitativa la carga de la CPU.

La RdP empleada presenta dos transiciones con tiempo que tendrán una *distribución exponencial,* por los siguientes motivos:

* + Se mantienen las propiedades de la red autónoma subyacente.
  + El marcado caracteriza al estado (por propiedad de la interpretación estocástica exponencial)

Como dijimos previamente, esta red modela una CPU que parte del estado de “stand by” (Stand\_by) y mueve a la CPU al estado de encendido (CPU\_ON) cuando las tareas son generadas.

La CPU se mantendrá en estado “encendido” tanto como trabajos existan en el Buffer del CPU. Si no hay tareas en el buffer, después de un intervalo de tiempo, la CPU se moverá nuevamente al estado “stand by” para conservar energía.

El modelo usa un generador de tareas abierto porque cuando la transición T1 dispare y deposite una tarea en el CPU\_Buffer, un token se mueve hacia la plaza P0a y habilita nuevamente a la transición que permite la generación de una nueva tarea.

Simulamos la CPU de doble núcleo por medio de la PN siguiendo los pasos a continuación:

1. Las tareas se generan en la plaza P0b cuando la transición “Arrival\_Rate” dispare aleatoriamente en el intervalo [0,1] usando la distribución exponencial. Y el token de la Plaza P0a, se deposita en P0b y habilita las transiciones T1\_1 y T2\_1.
2. Las transiciones T1\_1 y T2\_1 son transiciones inmediatas, y dispararan tan pronto como les sea posible. Estas tendrán la mayor prioridad de entre todas las transiciones, si existen más sensibilizadas a la vez. Cuando T1\_1 o T2\_1 disparen, el token es removido de la plaza P0b y depositadas en el buffer elegido, en la plaza inicial P0a y en la plaza P1\_2 o P2\_2. Inicialmente, la CPU está en modo Stand\_by, y cuando la tarea arriba, y el token es depositado en estas plazas, las transiciones T1\_3 o T2\_3 pasan a estar habilitadas.
3. Suponiendo el recorrido de uno de los núcleos, cuando T1\_3 dispare, los dos tokens de Stand\_by1 y de P1\_2 son removidos y dos tokens son generados. Uno se ubica en la plaza Power\_Up1, y otro es depositado nuevamente en P1\_2. Así la transición de Power\_Up\_Delay1 pasa a estar sensibilizada.
4. Como Power\_Up\_Delay1 es una transición determinista, la misma dispara después de un intervalo fijo, y un token es depositado en la plaza de CPU\_ON. Ahora sí la CPU pasa a estar encendida y lista para procesar el evento que llegó antes.
5. El token depositado en CPU\_Buffer1, el de CPU\_ON y el depositado en la plaza Idle1 habilitan la transición T1\_4. Y cuando esta dispare, el sistema pasa a estar en modo de procesamiento. Y con un token en Active1, la transición Service\_Rate es sensibilizada.
6. Luego de que Service\_Rate dispare, el token se quita de la plaza Active1 y se deposita un token en la plaza Idle1.
7. En el caso de que otra tarea arribe mientras el sistema está encendido, y procesando otra tarea, un token va a ser depositado en P1\_2 y en CPU\_Buffer1. Cuando la CPU esté encendida, la transición T1\_2 se disparará apenas esté habilitada, pero la tarea permanecerá en el buffer, ya que la exclusión la mantiene la tarea que está siendo procesada. Las tareas se acumularán en los buffer (hasta ahora no se define capacidad)
8. En el caso que el Arrival\_Rate de tareas sea muy lento para alguno de los núcleos, la CPU pasará al estado de Stand\_By1 luego de que la transición Power\_Down\_Threshold1 se sensibilice por no tener tokens en las plazas Active1 y CPU\_Buffer1. Este tiempo también será fijo, por ser una transición determinista.

Para la simulación de transiciones temporales y deterministas, se tomaran los siguientes parámetros:

|  |  |  |
| --- | --- | --- |
| Arrival\_Rate | 0.1 eventos per sec | 1 evento arriba cada 10 segundos |
| Service\_rate | 0.2 eventos per sec | 1 evento atendido cada 5 segundos |
| Power\_Down\_Threshold | 0.1 eventos per sec | Se apaga después de 10 segundos |
| Power\_Up\_Delay | 0.3 eventos per sec | Se activa después de 3 segundos |
| Tiempo total estimado para 1000 eventos | FALTA segundos | |

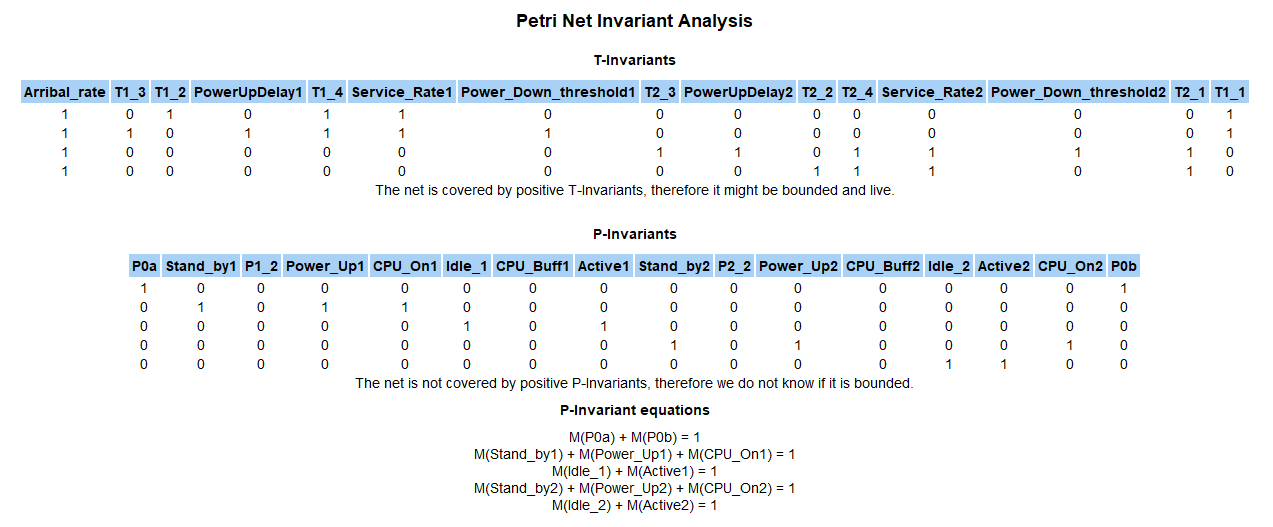
*Matrices y vectores empleados:*

* Matriz de incidencia I:
* Matriz de brazos inhibidores H:
* Matriz de intervalos temporales “Timer”:
* Vector de inhibidos por el tiempo Z:
* Vector de marcado Mj:
* Vector de transiciones sensibilizadas E:
* Vector de transiciones inhibidas por arco inhibidor B:
* Vector de sensibilizado extendido Ex:
* Vector de prioridades:

*Threads empleados:*

* + - Thread1: que se encarga del arribo de tareas y de elegir cuál núcleo/buffer emplear.
    - Thread2 y Thread3: los que se encargan de levantar del modo Stand\_By a la CPU, y de disparar a la transición T1\_2
    - Thread4 y Thread5: los que se encargan de atender a la tarea.
    - Thread6 y Thread7: los que se encargan de poner a la CPU en el modo Stand\_By nuevamente.

**Análisis de invariantes de la Red de Petri**



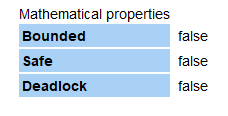
Las invariantes de transición y de plaza son útiles para probar propiedades dinámicas de la red, como vivacidad, inanición, alcanzabilidad, etc.

Por medio del análisis de los T-Invariantes, se pueden observar los circuitos/caminos que tendrá la RdP. Éstas indican posibles loops en la red. Y por medio de t-invariantes positivos, la RdP puede ser segura y viva.

Por medio del análisis de los P-Invariantes, se puede advertir el número de tokens que se tendrá en todas las marcas alcanzables.

**Propiedades de la Red de Petri**

Al analizar la red, empleando el software “Petrinator”, vemos que la misma tiene las siguientes propiedades de alcanzabilidad, seguridad e inanición:



1. Conclusiones: