Implementación de un Monitor con una Red de Petri Ejecutable

Ms. Ing. Micolini Orlando[1] , Ing. Julio Pailler[2], Ing. Miguel Tejeda[3]

[1] omicolini@compuar.com ,Becario SeCyT UNC
[2] juliopailler@hotmail.com ,Becario LAC
[3] tj_cala@hotmail.com , Becario LAC

Abstract. En la implementación de un monitor hay diferentes aspectos, los que aquí se resuelven son los referentes a su estado y la lógica del mismo. Para lo cual se usa una red de Petri. Vemos cómo se facilita la expresión del estado del sistema, la administración de los recursos, las prioridades y la solución para el problema de la inanición haciendo uso de una red de Petri en la implementación de un monitor.

Keywords: Monitor, Sincronización, Variable de estado

1 Objetivos

El presente trabajo tiene por objetivos: mostrar cómo implementar la lógica de un monitor especificadas mediante una red de Petri la cual es el modelo del problema a resolver por el monitor. También, nos proponemos establecer las bases para una metodología que nos permita independizar la lógica de un monitor y sus políticas. Lo que nos permite reusar las políticas del monitor cambiando fácilmente su lógica, que en definitiva se trata de una matriz.

2 Marco Teórico

2.1 Monitores

Los monitores son mecanismo de abstracción de datos (representan recurso en forma abstracta) con variables que caracterizan el estado del recurso. Los monitores tienen como finalidad sincronizar y comunicar sistemas informáticos que cooperan haciendo uso de memoria compartida [1].

El programador, cuando hace uso de un monitor, solo se preocupa por el método y los argumentos que solicitan o retornan el recurso.

Si se ha comprobado el buen funcionamiento del monitor puede ser reusado.

Existen construcciones explicitas de monitores en distintos lenguajes [2] de programación.

Los monitores son un importante mecanismo de control de concurrencia en la actualidad y en el futuro previsible [1].

Existen diferentes tipos de monitores que han sido publicados y evaluados

Andrews y Schneider [4] han clasificado los monitores, como con bloqueo o sin bloqueo de variables y han descripto técnicas de programación apropiadas para cada uno de ellos (basado en la clasificación de Howard)

Los componentes de un Monitor son: el código de inicialización que establece el estado inicial de los recursos; las variables locales que almacenan el estado interno del recurso y el estado interno de algún procedimiento que son accedidas solamente desde el monitor; procedimientos internos que operan las variables locales; procedimientos que son exportados, los que son accedidos por procesos activos que acceden al monitor, estos procedimientos son los que nos permiten usar los recursos del monitor, y el control de la exclusión mutua que está basado en colas que se asocian al monitor, «colas del monitor» [5].

Gestión de las colas del Monitor: cuando un proceso activo ejecuta uno de los procedimientos exportados del monitor, se dice que «Está Adentro del Monitor», el acceso al monitor garantiza que sólo un procedimiento este en el monitor; si hay un procedimiento en el monitor y otro trata de ingresar, este último se bloquea en la cola del monitor, y cuando un procedimiento abandona el monitor, selecciona el que está primero en la cola y lo desbloquea, si no hay ninguno el monitor queda libre.

Condiciones de Sincronización de un Monitor: si un proceso está en el monitor y no obtiene el recurso no puede seguir su ejecución, por lo que se requiere un mecanismo que lo bloquee hasta que el recurso esté disponible y pueda ser desbloqueado; este mecanismo es implementado con variables de condición, estas son accesibles solo desde el monitor, y con estas variables se realiza la gestión de sincronización (bloqueo y desbloqueo de los procesos en las colas del monitor).

Operación Delay: si c es una variable de condición, delay(c) hace que el proceso que la ejecuta se bloquee; antes de ejecutar delay por una condición, se debe desbloquear la cola de espera, de otra forma el monitor queda bloqueado, y por lo que Delay(c) debe liberar la exclusión mutua del monitor y bloquear el proceso que la ejecuto.

Según lo dicho un monitor puede tener tres colas: de **entrada** al monitor, de **espera** por las condiciones (pueden ser más de una) y para el **proceso señalizador** del desbloqueo. Ahora podemos clasificar los monitores según la prioridad específica de cada cola, las que son la prioridad de entrada (EP), prioridad de espera (WP), y la prioridad de señalizador (SP).

En la clasificación realizada según las prioridades relativas de estas tres colas resultan 6 casos concretos, de otra forma es posible que el monitor genere inanición.

La tabla 1 muestra estos casos y los nombres que se dan en la columna de la derecha que se corresponden con de la clasificación de Howard [1].

Tabla 1. Muestra la clasificación de Haward de los monitores según la prioridad de la cola.

Prioridad Relativa	Nombre tradicional del Monitor
EP=WP=SP	
EP=WP<SP	Wait and Notify [Lampson and Redell 1980]
EP=SP<WP	Signal and Wait [Howard 1976a]
EP<WP=SP	
EP<WP<SP	Signal and Continue [Howard 1976b]
EP<SP<WP	Signal and Urgent Wait [Hoare 1974]

2.2 Redes de Petri

Una Red de Petri o Petri Net (PN) es un modelo gráfico, formal y abstracto para describir y analizar el flujo de información. Conforma una herramienta matemática que puede aplicarse especialmente a los sistemas paralelos que requieran simulación y modelado de concurrencia con recursos compartidos [6].

PN están asociadas con la teoría de grafos y se pueden considerar como autómatas formales y generadores de lenguajes formales.

Las PN están formadas por: Plazas o Lugares, Token, Transiciones, Arcos Dirigidos y Peso.

Una PN se compone de los siguientes elementos:

- $L = \{l_1, l_2, l_3 \dots l_m\}$, conjunto de n lugares con n finito y distinto de cero;
- $T = \{t_1, t_2, t_3 \dots t_n\}$, conjunto de m transiciones con m finito y distinto de cero;
- I^- , matriz de incidencia negativa. Esta matriz es de dimensiones mxn representa los pesos de los arcos que ingresan desde los lugares de L a las transiciones de T;
- I^+, matriz de incidencia positiva. Esta matriz es de dimensiones mxn representa los pesos de los arcos que salen desde las transiciones de T hacia los lugares de L;

A partir de las dos últimas definiciones, se obtiene la **Matriz de Incidencia** $I = I^+ - I^-$. Para una red con m lugares y n transiciones, la misma es una matriz mxn cuyos elementos a_{ij} son:

$\forall\ l_i \in L$ y $\forall\ t_j \in T$ se tiene

- $a_{ij} = 0$, si entre l_i y t_j no hay arcos que los relacionen;
- $a_{ij} = W_{ij}$, si l_i es salida de t_j;
- $a_{ij} = -W_{ij}$, si l_i es entrada a t_j.

Con W_{ij} peso del arco que une l_i con t_j.

Marcado, es la distribución de los tokens en los lugares. Se corresponde con estados específicos de la red

Considerando los conceptos expuestos, una Red de Petri queda definida como una 5-tupla $PN = (L, T, I^-, I^+, m_0)$, donde m_0 es el marcado inicial de la red.

Se utilizan las siguientes definiciones, siendo F el conjunto de todos los arcos:
• t= conjunto de lugares de entrada a t;
t• = conjunto de lugares de salidas de t;
• l= conjunto de transiciones de entrada de l;
l• = conjunto de transiciones de salida a l.

- **Transición preparada:** se dice que t_i lo está si y sólo si todos sus lugares de entrada tienen al menos la cantidad de tokens igual al peso de los arcos que los une a t_i. Esto es $\forall l_i \in I^-(t_j) \Rightarrow m(l_i) \geq W_{ij}$, con W_{ij} peso del arco que une l_i con t_j. Si dos o más transiciones se encuentran preparadas en un mismo instante, se dice que están preparadas concurrentemente.

- **Disparo de una transición:** ocasiona el cambio de estado de una red, es decir el cambio del marcado de la misma.
  La función de disparo ∂ para una transición preparada t_i establece:

$$\partial(m_k, t_j) = \begin{cases} m_{k+1}(l_i) = m_k(l_i) - W_{ij} & \forall l_i \in \bullet t_j; \\ m_{k+1}(l_i) = m_k(l_i) + W_{ij} & \forall l_i \in t_j \bullet; \\ m_{k+1}(l_i) = m_k(l_i) \quad en \quad otro \quad caso. \end{cases}$$

- **Ejecución de una PN**: es la secuencia de pasos que resultan de disparar la red n veces partiendo desde el marcado inicial m_0, la que puede representarse mediante dos sucesiones:
  - Una secuencia de marcados por los que pasa la red en forma secuencial (m_0, m_1,…,m_n);
  - Una sucesión de transiciones que se van disparando (t_0, t_1,…, t_n) tales que ∂ (m_k, t_i) = m_{k+1}.

La ejecución de una red no es única, dando así una característica de no determinismo.

- **Ecuación de Estado:** la ecuación de estado de una red de Petri (que explicita el estado de la red en cada instante), queda definida en función de la matriz de incidencia I, y de un vector de disparo u_i de dimensión 1xn (siendo n la cantidad de transiciones) cuyas componentes son todas nulas, excepto la que corresponde a la transición disparada en el instante i, que vale 1 puesto que se trata de un solo disparo en una sola transición:

$$m_{i+1} = m_i + I \cdot u_i$$

Se deduce que el marcado final de una secuencia de disparos (partiendo de del marcado inicial m_0), puede obtenerse aplicando sucesivamente la ecuación de estado, quedando así lo siguiente:

$$m_i = m_0 + I \cdot \sum_{j=1}^{i} u_j = m_0 + I \cdot \bar{s}$$

Con $\bar{s} = \sum_{j=1}^{i} u_j$ vector asociado a la secuencia de disparo de las transiciones de la red; donde su j-ésima componente es igual al número de veces que la transición tj se ha disparado.

3 Uso de redes de Petri en la construcción de Monitores

Podemos ver a un monitor como dividido en dos secciones, las que son: primero la referida a la política de colas que se debe ejecutar para lograr que sólo un proceso esté en el monitor, que se bloqueen los procesos que no tienen los recursos y que se desbloqueen los que obtuvieron los recursos, y segunda la lógica con que se administran los recursos.

Supongamos un productor consumidor como el representado por la red de Petri de la figura 1 , la matriz de Incidencia y el vector de estado correspondiente, son los mostrados en las siguientes tablas.

,

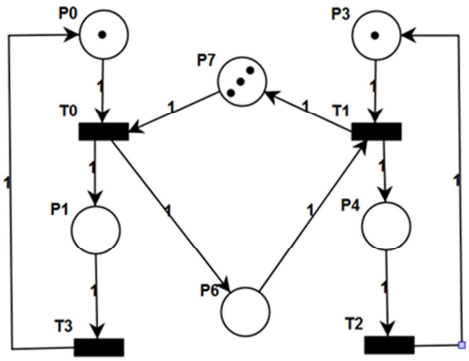

Figura 1. PN de un productor consumidor

Ahora cada vez que el productor quiere insertar un elemento hay que disparar la transición T0, para que el consumidor pueda sacarlo hay que disparar la transición T1, una vez disparado T0 o T1 hay que dejar el sistema preparado, es decir que las transiciones T0 y T1 estén preparadas para ejecutar un disparo, con lo que hay que disparar T3 o T2 según sea T1 o T2 el disparo anterior. Esto último se puede realizar explícitamente o sistólicamente.

	T0	T1	T2	T3
P0	-1	0	0	1
P1	1	0	0	-1
P3	0	-1	1	0
P4	0	1	-1	0
P6	1	-1	0	0
P7	-1	1	0	0

Tabla 2. Matriz de Incidencia de la PN de la figura 1.

Estado	P0	P1	P3	P4	P6	P7
Inicial	1	0	1	0	0	3

Tabla 3. Vector de estado según la marca de la figura. 1

La PN realiza el trabajo de la lógica del monitor, es decir la exclusión mutua y/o la administración de recursos disponibles; esto es "cuando el vector de estado que resulto del disparo no tiene componentes negativas es porque el sistema esta sincronizado o los recursos esta utilizable, de otro modo el proceso debe ir a una cola hasta que el recurso o la sincronización esté disponible".

Además debemos hacer notar que el vector de estado nos indica si el disparo ha devuelto o tomado recursos, en el ejemplo cuando el valor en P7, ver tabla 3, (cantidad de token) disminuye es porque se ha insertado un elemento en el buffer y cuando aumenta es porque se ha sacado un elemento.

Ahora bien, como podemos ver independientemente de la políticas implementadas en las colas del monitor la PN administra los recursos y la sincronización del sistema, puesto la PN es un modelo del sistema y lo que aquí se ha hecho es ejecutarla y evaluar el resultado con el fin de determinar si es posible la continuación de la ejecución del proceso o hilo que solicita el recurso o la sincronización.

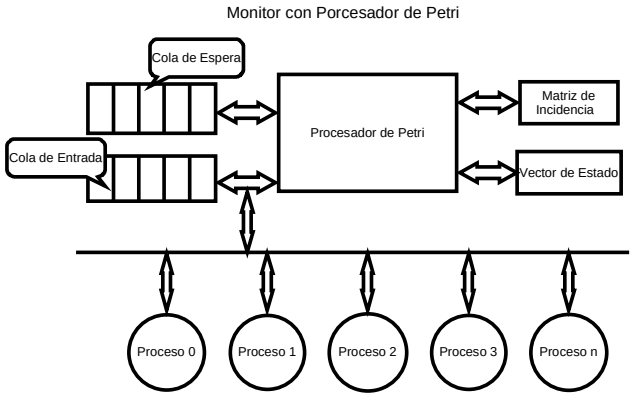

Figura 2. Modelo de un monitor con un Procesador de Petri

En la figura se ha representado un monitor con un procesador de Petri, el que consta de: la cola de espera de entrada (PE), la cola de espera por los recursos (W), los procesos que hacen uso del monitor, el procesador de Petri, la matriz de incidencia y el vector de estado.

La responsabilidad del procesador de Petri es calcular un estado después de un disparo para lo que realizar el siguiente cálculo.

$$m_i = m_0 + I \cdot \sum_{j=1}^{i} u_j = m_0 + I \cdot \bar{s}$$

Si el vector de disparo (s) solo realiza un disparo de una transición (u), todos sus componentes son cero excepto la del disparo que es uno, por lo cual no es necesario hacer la multiplicación con la matriz puesto que solo queda una columna, la del disparo, que se debe sumar con m_0 para obtener el estado siguiente y luego evaluar

que ninguna componente sea negativa para verificar que el estado es permitido o sea una transición permitida.

Cuando lo disparos se realizan de uno a la vez podemos también tener calculado el éxito de los disparos cono posibles para dar una respuesta inmediata a la solicitud de un recurso, esto es muy importante cuando se implementa por hardware puesto que mejora notablemente la latencia.

Para cambiar la lógica en esta solución solo es necesario cambiar la matriz de incidencia.

Para implementar una política de prioridades en la solución solo hay que colocar las prioridades en el mismo orden que los **lugares** de la PN, y cuando se recorreré la cola de los disparos no resueltos, que esperan en la cola de espera, si se lo hace por el índice del disparo también se lo hace en ese orden que en se han definido es la prioridad; además pueden ser implementadas otras prioridades y todo se reduce a la evaluación de la secuencia de disparos que están en la cola y los recursos que se dispone, lo cual puede determinarse evaluando el vector de estado.

4 Trabajo realizados con el Procesador de Petri

Para validar lo expuesto se han realizado numerosas implementaciones de la solución aquí propuesta, hay dos casos que se destacan por la diferencia en los dominios del problema y las herramientas para implementar las soluciones; estos casos son:

- Un monitor implementado en Java, aquí se dio solución al problema de la cena de los filósofos; múltiples productores y consumidores, y múltiples lectores y escritores .
- Un monitor por hardware, realizado en una FPGA Spartan 3E-1600 [7] y programado en VHDL haciendo uso de menos del 5% de los recursos, que hace uso del Procesador de Peti para la sincronización y gestión de recursos, resolviendo los mismos problemas antes mencionados.

En ambos caso lo único que se cambió en el código es la matriz de Incidencia, la que fue transferida como argumento al instanciar el monitor.

El proceso empleado para el desarrollo de soluciones es el siguiente:

- Elegir la política de colas con el objetivo de evitar la inanición y lograr las prioridades que se requieran.
- Modelar el problema con una PN
- Simular el problema haciendo uso de un simulador hasta validar que los problemas de gestión de recursos y sincronización están resueltos.
- Pasar como parámetro la matriz de incidencia al instanciar el monitor
- Ejecutar la PN con un método exportado por el monitor.

Esta metodología ha sido usada en los últimos 4 años para la solución de problemas en los cursos de Programación Concurrente en la carrera de Ing. en Comparación de la FCEFyN de la UNC. Los resultados obtenidos han sido una disminución del 70% de los errores en los trabajos presentados por los alumnos y una disminución en los tiempos de implementación del 30%; esto para grupos de distinto años lectivos.

5 Conclusiones

Hay que destacar que el aporte de este trabajo es la ejecución de una PN como la porción de código, que calcula y valida el nuevo estado de un sistema y que además tiene la posibilidad de determinar si hay bloqueo o inanición.

Como así también hacer uso de una herramienta grafica con un fuerte fundamento matemático (PN), que nos permite diseñar y validar el modelo de la solución que nos asegura que nuestro código es el modelo puesto que la matriz de Incidencia y el modelo representado por la PN son lo mismo.

6 Trabajos Futuros

Se encuentra en curso la integración del procesador de Petri a un sistema multi-core para ser usado como un coprocesador de concurrencia desde un lenguaje de alto nivel.

REFERENCIAS

[1] PETER A. BUHR AND MICHEL FORTIER, MICHAEL H. COFFIN, Vol. 27, No. 1, March 1995, Monitor Classification, ACM Computing Surveys

[2] BRINCH HANSEN, P. 1975. The programming language, concurrent pascal. Eng, 2 (June), 199—206.

[3] Programación Concurrente, Palma Mendez, Jose Tomas, Editorial Paraninfo, ISBN: 8497321847, 2003

[4] Andrews, G. R., and Schneider, F. B. 1983. Concepts and notations for concurrent programming. ACM Cornput. Suru. 15, 1 (Mar.),
3-43.

[5] Andrews, G. R. 1991. Concurrent programming: Principles and Practice. Benjamin / Cummings, Redwood City, Calif.

[6] Michel Diaz, Petri Nets Fundamental Models, Verification and Applications, Weley, 2009

[7] Spartan 3E-1600 Development Board, http://www.digilentinc.com