ÍNDICE

INTRODUCCIÓN

Este trabajo consta de la simulación de lo que ocurre en una esquina cuando varios autos de manera concurrente quieren cruzar la misma en distintas direcciones. Para esta simulación los autos van a ser representados como procesos (hilos de ejecución) y la esquina va a ser representada con un monitor, en el que la lógica del mismo será implementada mediante una Red de Petri (RdP).

Se realizarán una serie de testeos al monitor y luego se hará una conclusión sobre el resultado de los mismos.

MARCO TEÓRICO

Monitor

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.
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».
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.

Red de Petri (RdP)

Una Red de Petri o Petri Net (RDP) 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.
RDP están asociadas con la teoría de grafos y se pueden considerar como autómatas formales y generadores de lenguajes formales. Las RDP están formadas por: Plazas o Lugares, Token, Transiciones, Arcos Dirigidos y Peso.
Una RDP se compone de los siguientes elementos:
· L = {l1, l2, l3… lm}, conjunto de m lugares con m finito y distinto de cero;
· T = {t1, t2, t3… tn}, conjunto de n transiciones con n 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 aij son:
" li Î L y " tj Î T se tiene
· aij = 0, si entre li y tj no hay arcos que los relacionen;
· aij = Wij, si li es salida de tj;
· aij = -Wij, si li es entrada a tj.
Con Wij peso del arco que une li con tj.
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 RDP = (L, T, I-, I+, m0), donde m0 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 ti 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 ti. Esto es *i* (*j*) (*i*) *ij* "*l* Î*I* - *t* ⇒*m l* ³*W* , con Wij peso del arco que une li con tj. 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.

Ejecución de una RDP: es la secuencia de pasos que resultan de disparar la red n veces partiendo desde el marcado inicial m0, la que puede representarse mediante dos sucesiones:
-Una secuencia de marcados por los que pasa la red en forma secuencial (m0, m1,…,mn);
-Una sucesión de transiciones que se van disparando (t0, t1,…, tn) tales que ¶ (mk, ti) = mk+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 ui 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:
mi+1 = mi + I · ui.
Se deduce que el marcado final de una secuencia de disparos (partiendo de del marcado inicial m0), puede obtenerse aplicando sucesivamente la ecuación de estado, quedando así lo siguiente:

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

Donde *s* es un 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.

DESARROLLO

IMPLEMENTACIÓN DEL MONITOR CON UNA RED DE PETRI

Se puede 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.
En el monitor que aquí se ha desarrollado se implementó una política de prioridades en la cual se atienden los hilos dormidos en las variables de condición siguiendo el método de Round Robin.

ELEMENTOS INTERVINIENTES EN EL MONITOR

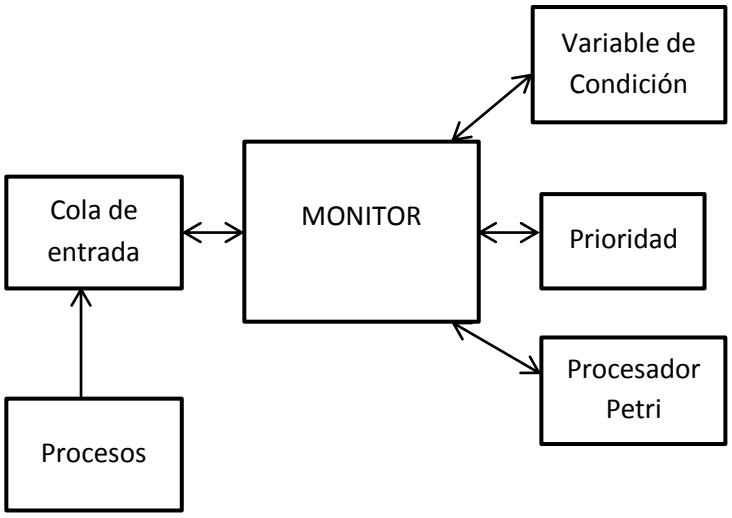

Figura 1. Diagrama que ilustra la interacción del monitor con los distintos elementos que lo constituyen

Cola de entrada

En la cola de entrada residen los procesos que quieren ejecutar un método exportado del monitor (es decir, que quieren entrar al monitor); esta cola de entrada será implementada por un semáforo binario con una política FIFO.

Variable de Condición

En las variables de condición van a residir los procesos que no hayan podido disparar una transición a causa de que no estaba disponible el recurso que necesitan; habrá tantas variables de condición como transiciones tenga la RdP con la que se está trabajando, y la identificación de cada variable de condición, se corresponderá con la identificación de la transición de la red, por ejemplo, la variable de condición 4 corresponderá a la transición T4 de la red.

Prioridad

La prioridad del monitor, la manejará un objeto independiente. En este caso se implementa con un array de enteros, el cual contendrá el número identificador de cada variable de condición; la política de prioridad será una política Round Robin. Para esclarecer esto, se da un ejemplo:

Suponga que se está trabajando con cuatro variables de condición, entonces el objeto prioridad en un comienzo será el siguiente

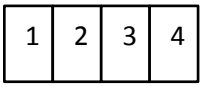

Figura 2. Elementos dentro del objeto "Prioridad"

Entonces cuando un hilo dentro del monitor empiece a revisar las variables de condición para ver si hay hilos dormidos dentro, empezara por la 1 seguirá por la 2 y 3, y la última que revisara será la variable de condición 4; suponga que encontró un hilo dormido y lo despertó en la VC1; pues entonces se modifica la prioridad para la próxima entrada de un hilo, y los elementos dentro dentro del objeto que maneja las prioridades del monitor será la siguiente:

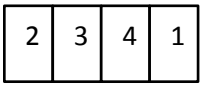

Figura 3. Elementos del objeto "Prioridad" después de haber despertado un hilo en la VC1

De modo que el siguiente hilo que entre al monitor, empezará revisando la VC2 y terminará por la VC1.

Procesador Petri

La RDP realiza el trabajo de la lógica del monitor, es decir la exclusión mutua y/o la administración de recursos disponibles; para hacer esto debe valerse de una Red de Petri, más bien, de la matriz de incidencia y marcado inicial de la misma.
El funcionamiento es el siguiente: el monitor pregunta a la RdP si se puede disparar una transición, entonces la red realiza el disparo de la transición solicitada y verifica la matriz de marcado que resultó de hacer el disparo, cuando el vector de estado que resultó 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".
Independientemente de las políticas implementadas en las colas del monitor la RdP administra los recursos y la sincronización del sistema, puesto la RdP es un modelo del sistema y lo que aquí se ha hace 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.

La red de Petri que representa a una esquina y que administra los recursos del monitor es la siguiente:

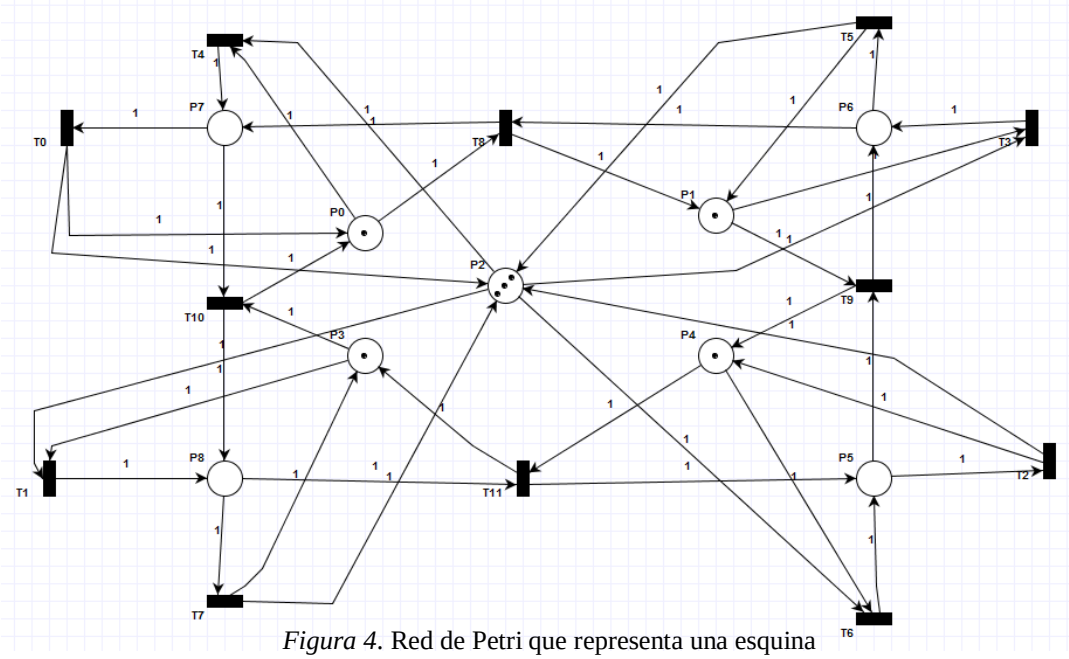

Figura 4. Red de Petri que representa una esquina

En las plazas P0, P1, P2, P3 y P4 se representan los recursos disponibles; los tres tokens de la plaza P2 son para limitar a un máximo de tres autos por esquina, para evitar el interbloqueo.

DIAGRAMAS UML

Diagrama de clases del sistema que muestra al monitor con todos los demás objetos que lo componen.

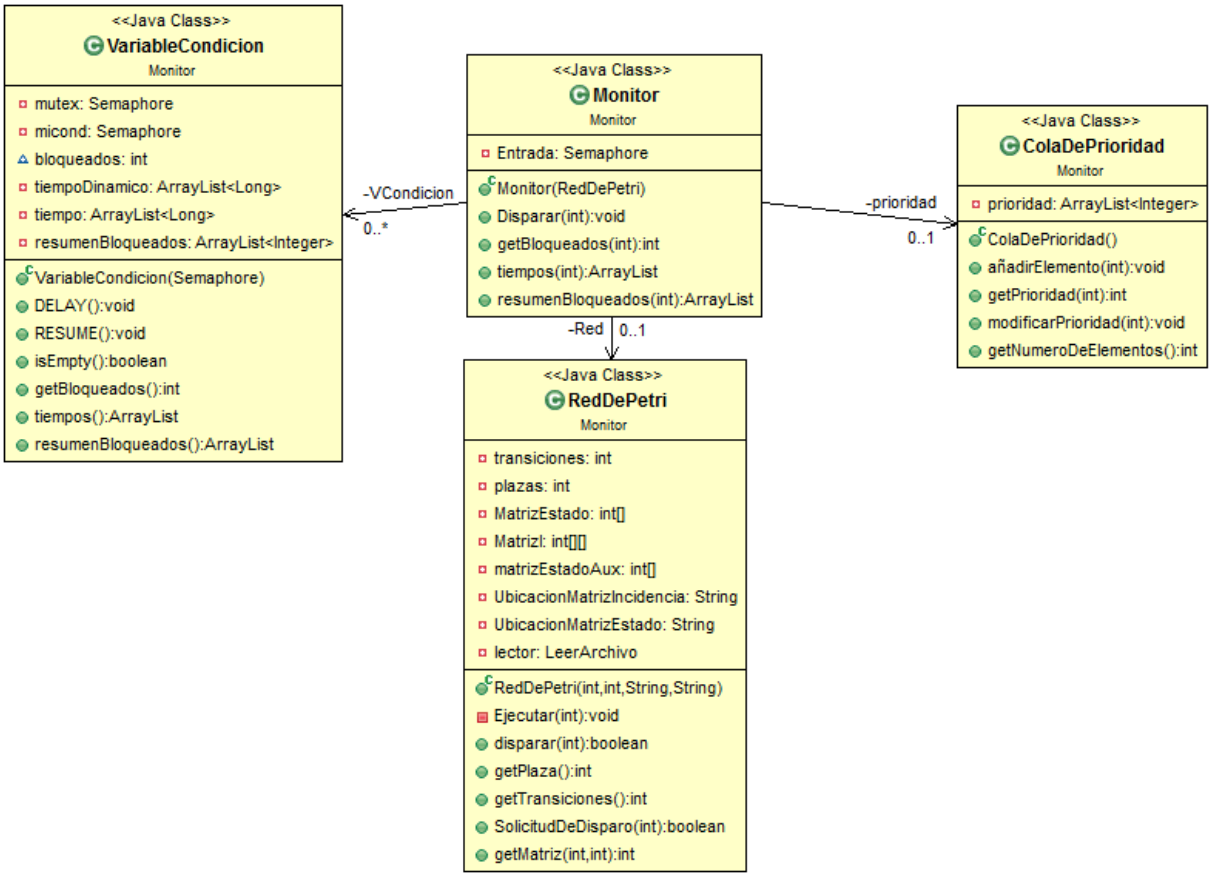

Figura 5. Diagrama de clases del monitor

Diagrama de secuencia que muestra la secuencia que ocurre cuando un auto ejecuta el método *disparar del monitor*

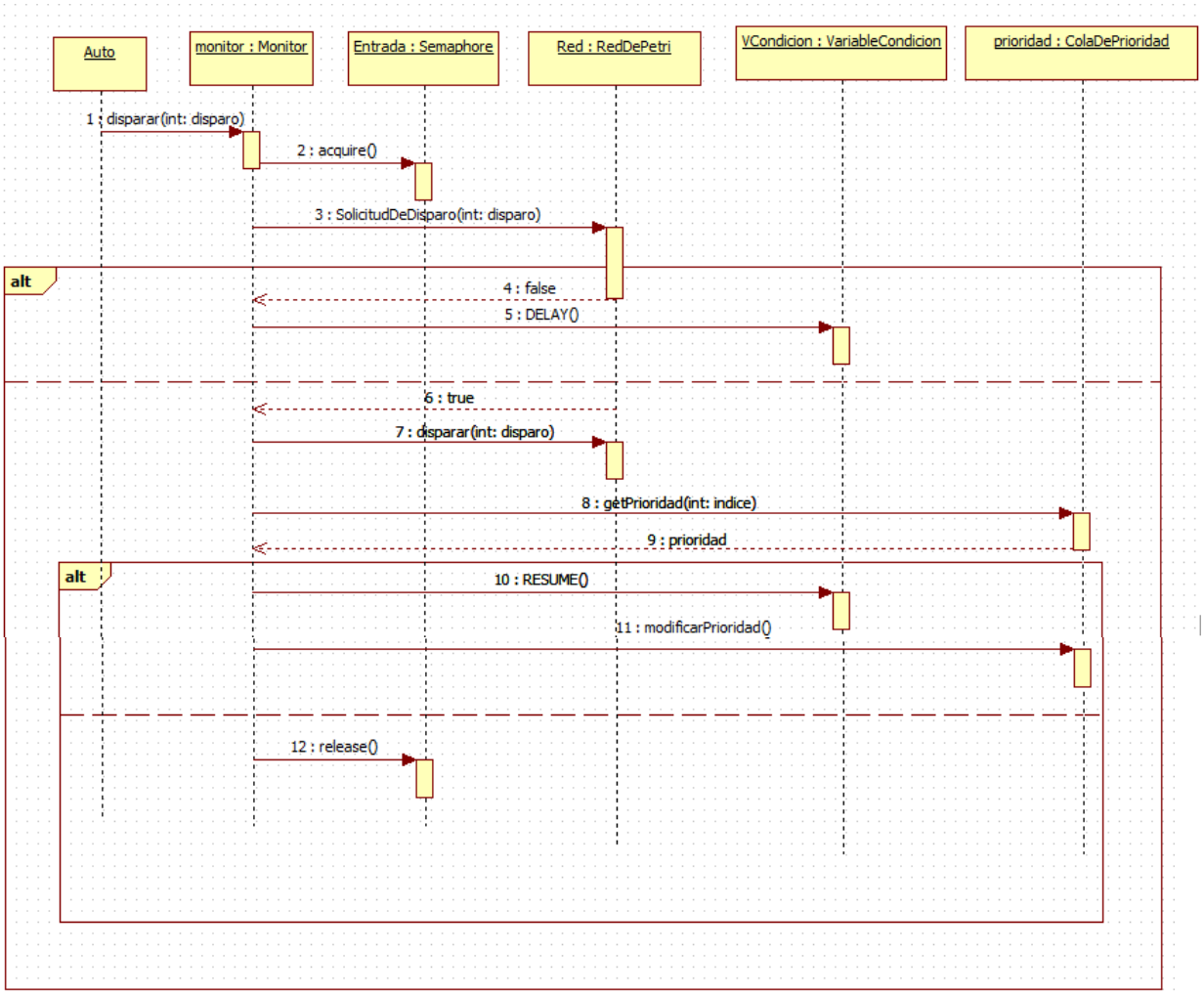

Figura 6. Diagrama de secuencia cuando un proceso ejecuta el método *disparar* del monitor

TESTING DEL MONITOR

Test Unitarios

Se realizaron dos test unitarios que se describen a continuación:

1)TestColaDePrioridad

Este test prueba la clase cola de prioridad, para comprobar el correcto funcionamiento de la misma, es decir, que se verifique el buen funcionamiento de la política Round Robin una vez que se ha cambiado la prioridad. Para ello, se crea un elemento cola de prioridad y se le agregan cuatro elementos enteros 0, 1, 2 y 3; luego se revisan las prioridades, se modifica la prioridad, y nuevamente se verifican las prioridades.

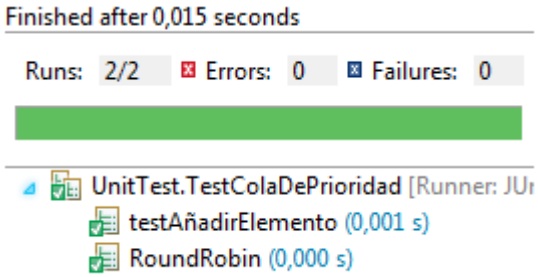

Figura 7. Resultado Test "TestColaDePrioridad"

2)TestMonitor

Este test comprueba el correcto funcionamiento de la lógica del monitor, se hacen dos pruebas:

Prueba Nº1

En esta prueba, dos autos intentan ejecutar la misma transición, una vez que el primer auto ejecuta la transición, ésta ya no va a estar sensibilizada y por lo tanto cuando el segundo hilo (auto2) intente ejecutarla, se bloqueará en la variable de condición correspondiente a la transición que intentó ejecutar.

Prueba Nº2

Esta es una prueba que verifica que no puede haber más de tres autos a la vez en una esquina para hacer esta prueba, se crearan 4 Autos (hilos) el tercer hilo que intente entrar en la esquina no lo va a poder hacer y por lo tanto se bloqueara en una VC. Para esta comprobación se verificara el estado de los hilos, todos debieron haber terminado su ejecución, salvo el hilo bloqueado.

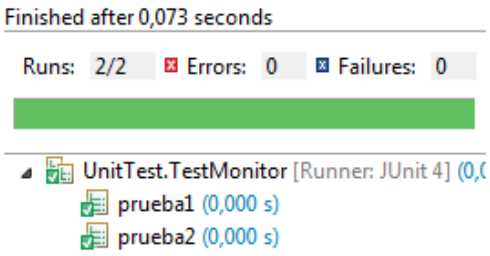

Figura 8. Resultado Test "TestMonitor"

Test de sistema

Se realizaron tres test de sistema, a continuación se describen cada uno y por último los gráficos que muestran el resultado obtenido luego de correr el test.

Al final se hará una conclusión sobre el resultado de los mismos.

Id Caso de Prueba	01	
Tipo de prueba	Sistema	
Objetivos de la prueba	Comprobar el correcto funcionamiento de la clase monitor	
Descripción	Es un test simétrico, es decir, se crea un número fijo e igual de autos que recorren todas las direcciones posibles de la esquina.	
Prerrequisitos de la prueba		
1.- Crear un objeto Monitor 2.-Crear un objeto RedDePetri 3.-Crear 200 autos (hilos) que recorran las doce direcciones posibles (200 por cada dirección posible)		
Procedimientos		
1.- Ejecutar la clase TestMonitorSimetrico		
Resultado esperado		
1.- Todos los hilos deben terminar su ejecución 2.- No hay más de un auto en las plazas del monitor 3.- No se forman colas en las transiciones que retornan recursos		
Resultados obtenidos		
1.- Los resultados obtenidos coinciden con el resultado esperado.		
Observaciones		
Resultado de la prueba	x	Aprobado
		No aprobado

Figura 9. Test de sistema 01

Figura 10. Tiempos máximos que esperaron los hilos en las variables de condición

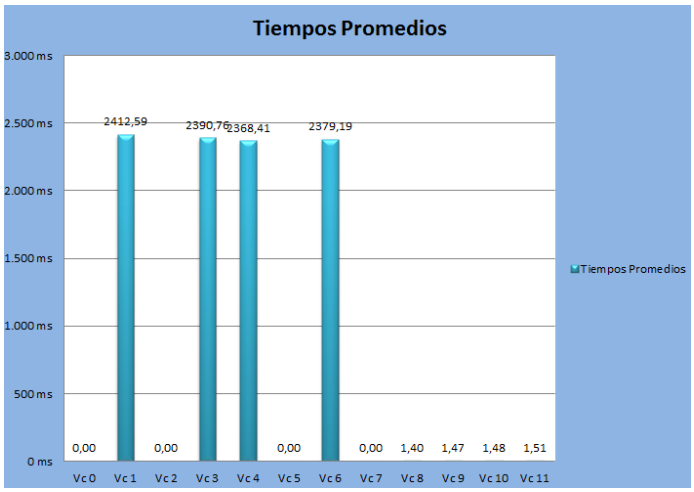

Figura 11. Tiempos promedio que esperaron los hilos en las variables de condición

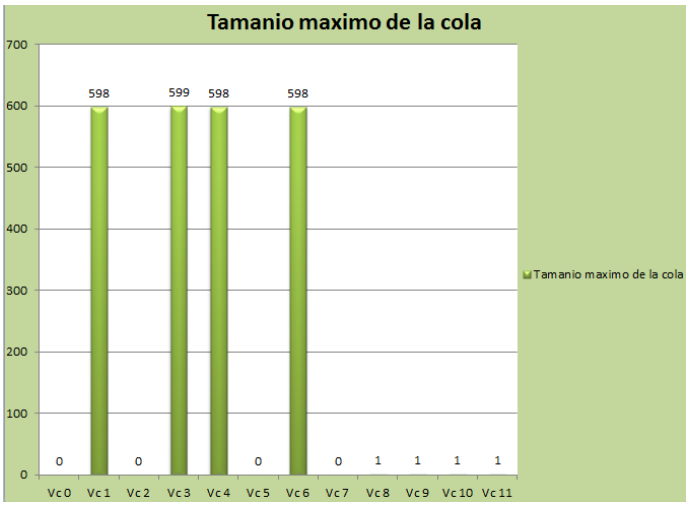

Figura 12. Tamaño máximo que alcanzaron las colas

Id Caso de Prueba	02	
Tipo de prueba	Sistema	
Objetivos de la prueba	Comprobar el correcto funcionamiento de la clase monitor	
Descripción	Es un test asimétrico, es decir, se crea un número fijo e igual de autos que recorren todas las direcciones posibles de la esquina excepto una dirección, para ésta dirección se crean 200 veces más autos que para las demás.	
Prerrequisitos de la prueba		
1.- Crear un objeto Monitor 2.-Crear un objeto RedDePetri 3.-Crear 200 autos (hilos) que recorran once direcciones posibles (200 por cada dirección) 3.- Crear 2000 autos (hilos) que recorran una dirección posible		
Procedimientos		
1.- Ejecutar la clase TestMonitorAsimetrico		
Resultado esperado		
1.- Todos los hilos deben terminar su ejecución 2.- No hay más de un auto en las plazas del monitor 3.- No se forman colas en las transiciones que retornan recursos		
Resultados obtenidos		
1.- Los resultados obtenidos coinciden con el resultado esperado.		
Observaciones		
Resultado de la prueba	x	Aprobado
		No aprobado

Figura 13. Test de sistema 02

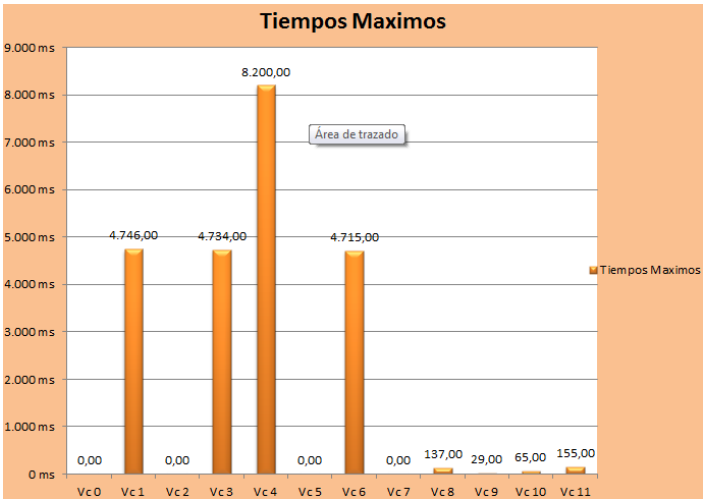

Figura 14. Tiempos máximos que esperaron los hilos en las variables de condición

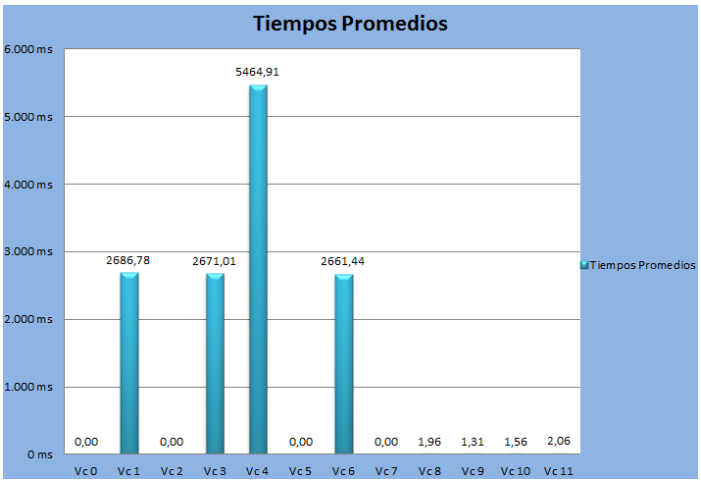

Figura 15. Tiempos promedio que esperaron los hilos en las variables de condición

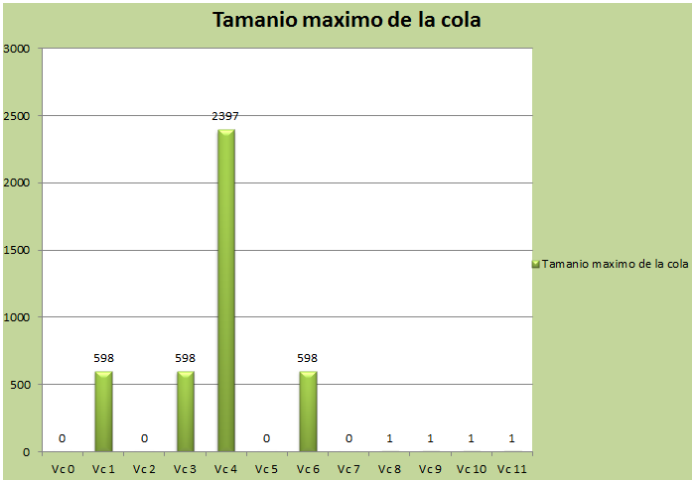

Figura 16. Tamaño máximo que alcanzaron las colas

Id Caso de Prueba		03
Tipo de prueba		Sistema
Objetivos de la prueba		Comprobar el correcto funcionamiento de la clase monitor
Descripción		Se crean autos en una única dirección con el propósito de comparar el tiempo de espera en las variable de condición respecto al test 01(autos en todas las direcciones)
Prerrequisitos de la prueba		
1.- Crear un objeto Monitor 2.-Crear un objeto RedDePetri 3.- Crear 600 autos (hilos) que recorran una dirección posible (Norte a Sur)		
Procedimientos		
1.- Ejecutar la clase TestMonitorNum3		
Resultado esperado		
1.- Todos los hilos deben terminar su ejecución 2.- No hay más de un auto en las plazas del monitor 3.- No se forman colas en las transiciones que retornan recursos 4.- No debe haber hilos bloqueados en la variable de condición 10		
Resultados obtenidos		
1.- Los resultados obtenidos coinciden con el resultado esperado.		
Observaciones		
1.- En algunos casos el test da como resultado un hilo bloqueado en la variable de condición 10, esto responde a cómo gestiona el microprocesador los hilos, es decir, nada asegura que el último hilo que disparó una transición va a ser el que primero adquiera la exclusión mutua		
Resultado de la prueba	x	Aprobado
		No aprobado

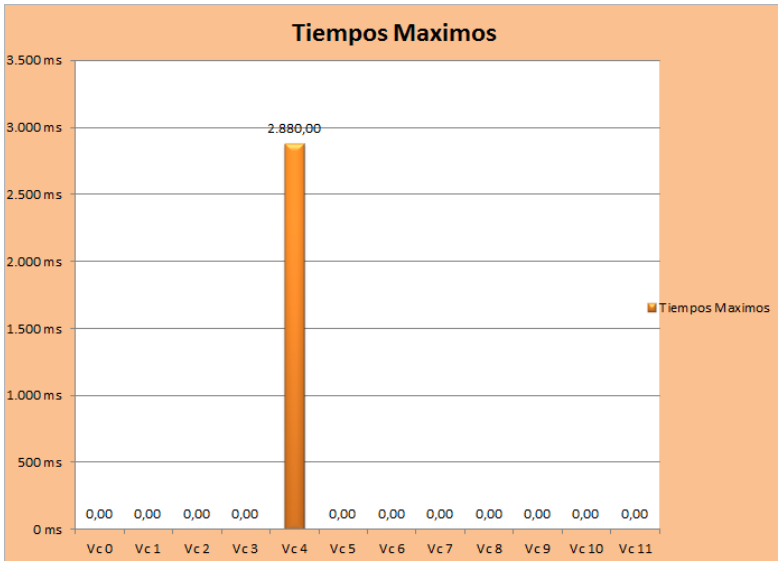

Figura 17. Tiempos máximos que esperaron los hilos en las variables de

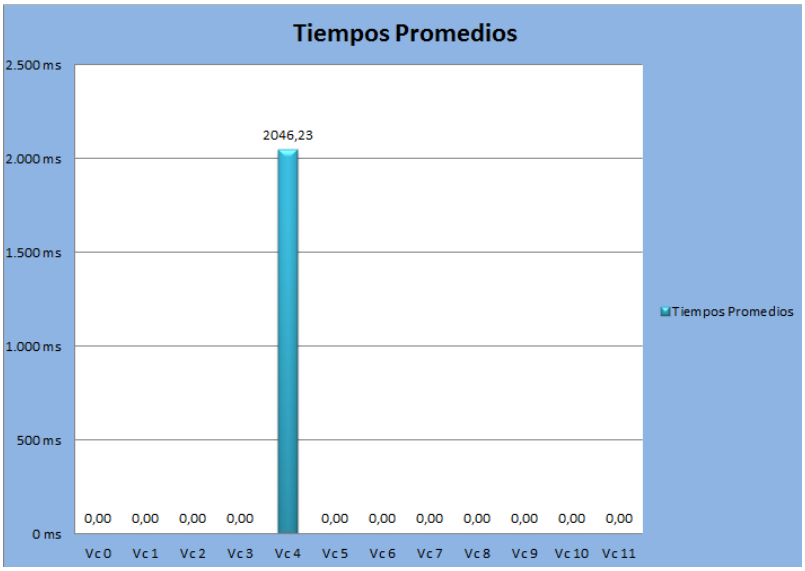

Figura 18. Tiempos promedio que esperaron los hilos en las variables de condición

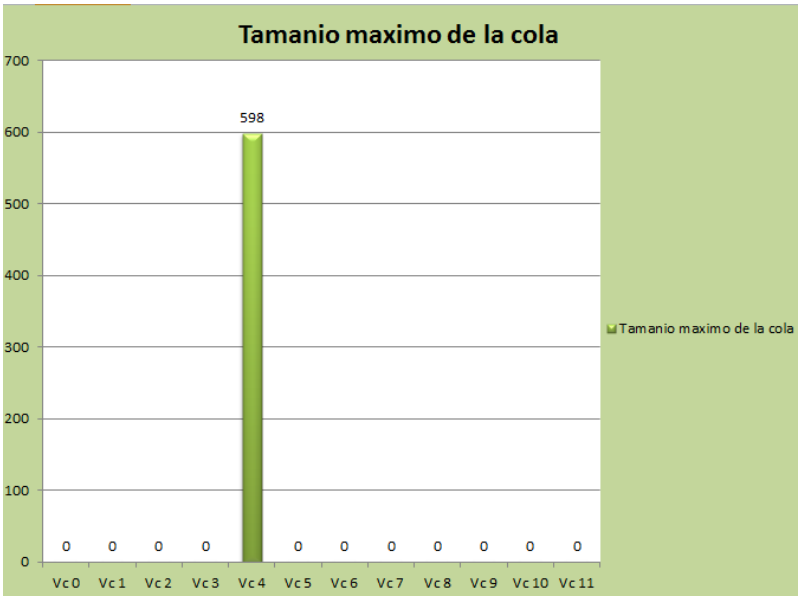

Figura 19. Tamaño máximo que alcanzaron las colas

CONCLUSIÓN

De acuerdo a los resultados de los test realizados concluimos que:

1) No hay dos autos en el mismo lugar dentro de la esquina

Se garantiza esto de acuerdo a los gráficos de los resultados de test, el tamaño de las colas de las variables de condición (VC8, VC9, VC10, VC11) no superan la unidad en ninguno de los casos; esto significa que no hay más de un auto en las plazas interiores.

2) No hay más de tres autos cruzando la esquina simultáneamente

Esto se puede comprobar con el UnitTest02, en el que, cuando un cuarto auto intenta ingresar a la esquina no puede hacerlo.

3) No hay interbloqueo

En todos los test de sistema se comprueba que todos los autos hayan terminado su ejecución, ya que si esto no sucede, el test nunca finalizaría.

4) No hay inanición

El tiempo en que un hilo está dormido en una variable de condición interna, no supera el tiempo que los hilos están dormidos en las variables de condición de entrada a la esquina.

5) No se asegura que el orden con el que los hilos salen del monitor, sea el mismo con el que se encolan en la entrada.

Se llega a esta conclusión de acuerdo a los resultados del test de sistema 03, ya que, si el orden con el que los hilos realizan los disparos y luego se encolan en la entrada del monitor se respetara, no se debería formar cola en la variable de condición 10 (VC10); esto sucede por como gestiona los procesos el procesador, es ajeno al monitor.

6) El tamaño máximo de las colas del monitor (ya sea la cola de entrada o las colas de las variables de condición) está sujeto a características del hardware en el que se está ejecutando el monitor

Como las colas del monitor están implementadas con la clase “semaphore” de la librería “concurrent” que proporciona la API de Java, la especificación de la misma establece que el tamaño máximo de la cola (es decir, los hilos dormidos en el semáforo) está sujeto a las características del hardware.

REFERENCIAS

Implementación de un Monitor con una Red de Petri Ejecutable- Ms. Ing. Micolini Orlando , Ing. Julio Pailler, Ing. Miguel Tejeda

Programación Concurrente – José Tomás Palma Mendez

Filminas de clases teóricas de Programación Concurrente- Ing. Micolini, Orlando