Sistemas Operativos

Primer Cuatrimestre de 2009

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo práctico final

Abstract

Simulación del Algortimo del Banquero, Simulación de Semáforos, Simulación de Elección de Coordinador mediante *Token Ring*

Palabras clave

Simulación, Algortimo del Banquero, Semáforos, *Token Ring*

Integrante	LU	Correo electrónico
Matías López y Rosenfeld	437/03	matiaslopez@gmail.com
Tomás Santiago Scally	886/03	horizontedesucesos@gmail.com

Índice

3. Algoritmo del Banquero

El Algoritmo del banquero en sistemas operativos es una forma de evitar el *deadlock*, propuesta por primera vez por Edsger Dijkstra. Es un acercamiento teórico para evitar los *deadlocks* en la planificación de recursos. Requiere conocer con anticipación los recursos que serán utilizados por todos los procesos. Esto último generalmente no puede ser satisfecho en la práctica. El algoritmo mantiene al sistema en un estado seguro. Un sistema se encuentra en un estado seguro si existe un orden en que pueden concederse las peticiones de recursos a todos los procesos, previniendo el *deadlock*. Los procesos piden recursos, y son complacidos siempre y cuando el sistema se mantenga en un estado seguro después de la concesión. De lo contrario, el proceso es suspendido hasta que otro proceso libere recursos suficientes. En términos más formales, un sistema se encuentra en un estado seguro si existe una secuencia segura. Una secuencia segura es una sucesión de procesos, $< P_1, \ldots, P_n >$, donde para un proceso P_i, el pedido de recursos puede ser satisfecho con los recursos disponibles sumados los recursos que están siendo utilizados por P_j, donde $j < i$. Si no hay suficientes recursos para el proceso P_i, debe esperar hasta que algún proceso P_j termine su ejecución y libere sus recursos. Recién entonces podrá P_i tomar los recursos necesarios, utilizarlos y terminar su ejecución. Al suceder esto, el proceso P_{i+1} puede tomar los recursos que necesite, y así sucesivamente. Si una secuencia de este tipo no existe, el sistema se dice que está en un estado inseguro, aunque esto no implica que esté bloqueado. Así, el uso de este tipo de algoritmo permite impedir el *deadlock*, pero supone una serie de restricciones:

- Se debe conocer la máxima demanda de recursos por anticipado.
- Los procesos deben ser independientes, es decir que puedan ser ejecutados en cualquier orden. Por lo tanto su ejecución no debe estar forzada por condiciones de sincronización.
- Debe haber un número fijo de recursos a utilizar y un número fijo de procesos.
- Los procesos no pueden finalizar mientras retengan recursos.

En este trabajo realizamos un simulador para el Algoritmo del Banquero. La idea principal de este simulador es mostrar paso a paso la ejecución del algoritmo para una instancia que el usuario provea. Asi el usuario podrá saber si al llegar el pedido de un proceso, satisfacer este pedido mantiene al sistema en un estado seguro. Recordemos que el algoritmo asume que el sistema inicialmente se encuentra en un estado seguro.

3.1. Ejemplo de funcionamiento

Mostraremos a continuación el ejercicio 7.c de la guía de ejercicios de la materia. Este ejemplo cuenta con cinco procesos y cuatro recursos.

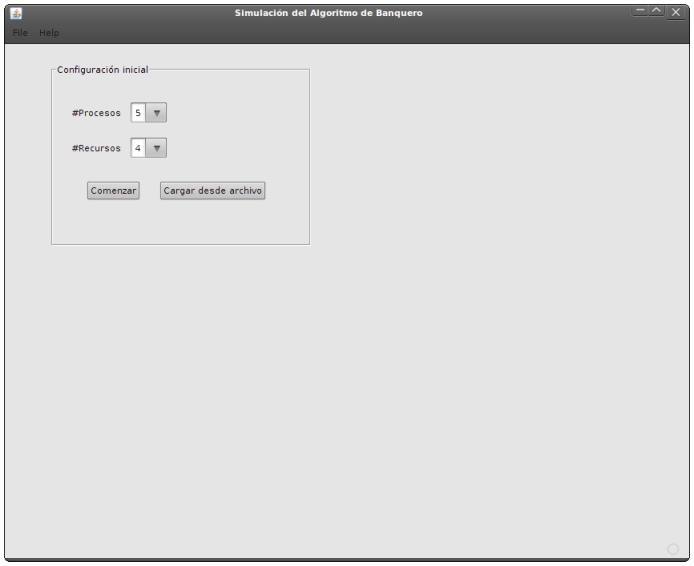

Figura 1: Esta es la primer pantalla del simulador, debemos escoger la cantidad de procesos y recursos o cargar una instancia guardada previamente desde el disco.

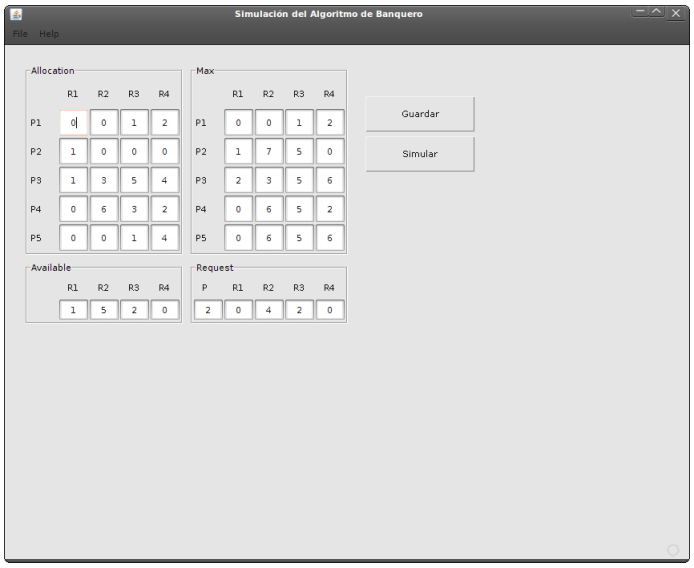

Figura 2: Aquí ya vemos la siguiente pantalla, en la que debemos ingresar los valores de entrada de la simulación. *Allocation* es la matriz que refleja los recursos que tiene asignado cada proceso, *Max* es el máximo de recursos que un proceso declara que va a usar, *Available* es la cantidad de recursos de cada tipo que están disponibles, *Request* muestra cuántos recursos de casa tipo pidió el proceso *P*. Después de ingresar los valores podemos optar por guardar los valores ingresados para ser usados luego, para esto utilizamos el botón *Guardar* o iniciar una simulación

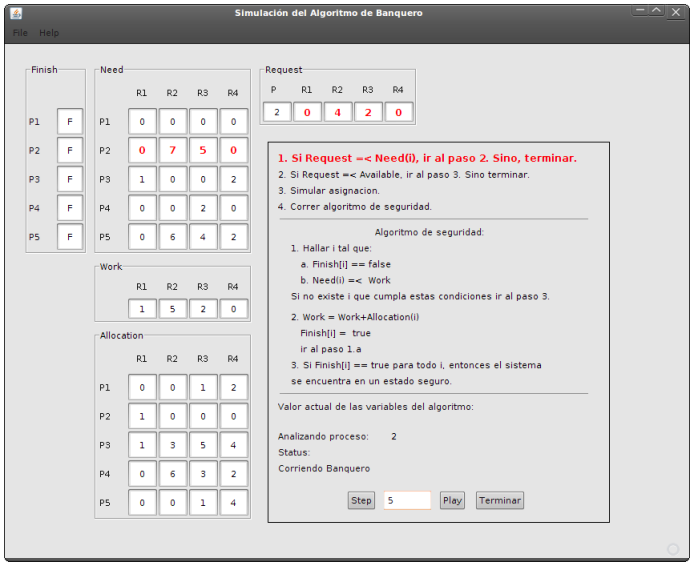

Figura 3: Esta es la última pantalla del simulador, en ella vemos los valores de las distintas matrices que participan en el algoritmo. El vector *Finish* nos indicará cuales de los procesos pueden terminar en la simulación. Si todo el vector *Finish* queda en *True*, querrá decir que el sistema está en un estado seguro. El botón *Step* avanza el algoritmo un paso, podemos presionarlo repetidas veces hasta finalizar la simulación, también podemos elegir el intervalo de tiempo entre paso y paso y presionar *Play*. En rojo se mostrará el paso del algoritmo que se está por ejecutar y los valores que se están utilizando en ese paso.

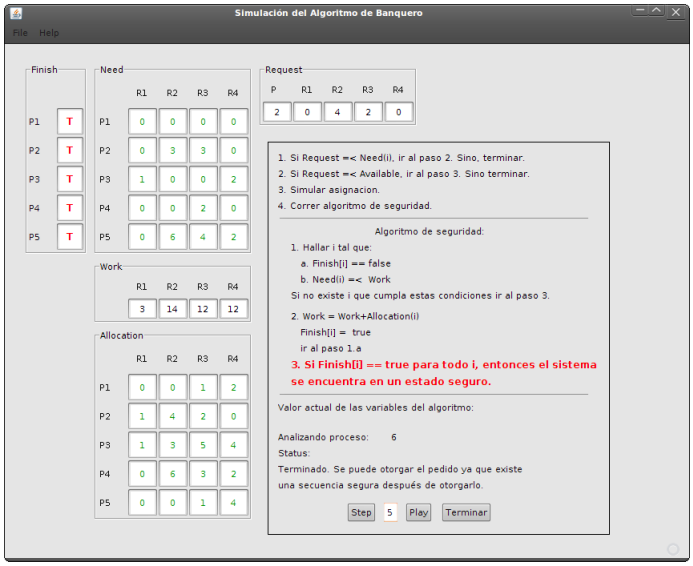

Figura 4: Aquí ya ha terminado la simulación, podemos ver el vector de *Finish* completamente *seteado* en *True*, eso nos indica que existe una secuencia segura en este sistema.

4. Elección de Coordinador en un Sistema Distribuido con *Token Ring*

En este simulador se puede observar cómo es llevada acabo la elección de un nuevo coordinador ante la caida del coordinador en un sistema distribuido mediante *Token Ring*.

Decidimos suponer una red de ocho computadoras interconectadas que tienen como coordinador al nodo de mayor número que esta activo. Creemos que ocho computadoras es un número suficientemente grande para ilustrar el algoritmo sin pérdida de generalidad.

La simulación permite poner *online/offline* los distintos nodos de la red y a los activos les permite verificar que el coordinador esté vivo. Esto se hace enviando un mensaje AYA (*Are you alive*), mensaje que en el coordinador debería responder con un IAA (*I am alive*).

Cuando un nodo detecta que el coordinador actual no está vivo, dado que no respondio el AYA que le envió (ocurre un *timeout* en la espera de la respusta), arma un mensaje que contendrá una lista en la cual se irán anotando los nodos activos. Inicialmente se incluye a si mismo en esta lista y la envia a través de la red a la siguiente computadora (recordemos que cada computadora tiene un número identificatorio). Si este nodo está activo, recibe el mensaje, se agrega a la lista y repite el procedimiento con su sucesor. En caso contrario el nodo emisor le envía el mensaje con la lista de nodos a la siguiente computadora repitiéndose el proceso anterior (si está activo se agrega y sino se reenvía al siguiente). Una vez que la lista viaja por toda la red dando una vuelta entera al anillo vuelve al nodo emisor, este inspecciona la lista para elegir al nuevo coordinador. El nodo que reemplazará al coordinador caido es el que tenga el número más grande de la lista. El nodo que comenzo el proceso de selección avisa al resto de los nodos que el nuevo coordinar ha sido elegido y envía el número del mismo.

4.1. Ejemplo de funcionamiento

Veamos ahora un ejemplo en el simulador:

Supongamos que tenemos una red de ocho computadoras numeradas del cero al siete. El nodo siete es el coordinador, pero por alguna razón se cae. El nodo número tres realiza un AYA al nodo coordinador para saber si esta activo, al no responder luego de un lapso de tiempo se produce un *timeout*. El nodo número tres arma la lista de nodos activos, se incluye en ésta y la envía al siguiente nodo, en este caso será el número cuatro, que se incluirá y asi sucesivamente hasta llegar al número 6. En este caso el nodo número 6 no recibe respuesta del nodo número siete y entonces lo saltea, enviándole la lista al número cero. Siguiendo este esquema la lista llegará nuevamente al número tres, que es el que inició el proceso de elección con lo cual sabrá que debe realizar la selección del nuevo coordinador entre los nodos que figuran en el mensaje. Para esto toma el mayor valor de la lista, resultando este el número seis. Le envia un mensaje al nodo número seis para que comience a coordinar a la red y avisa a los demás.

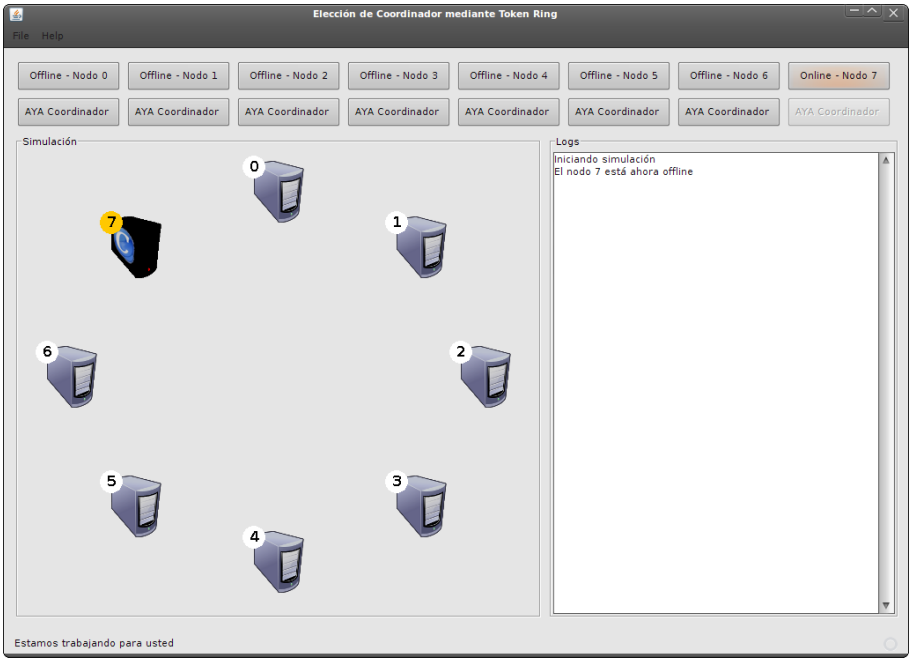

Figura 5: Aquí vemos al simulador con el nodo siete caido.

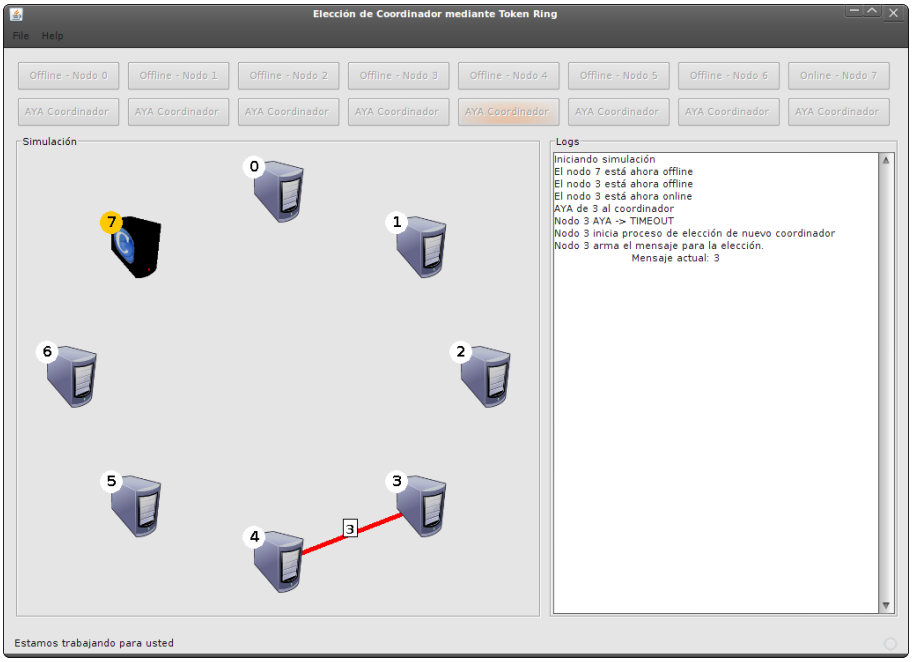

Figura 6: El nodo tres no recibe respuesta del coordinador y comienza el proceso de elección.

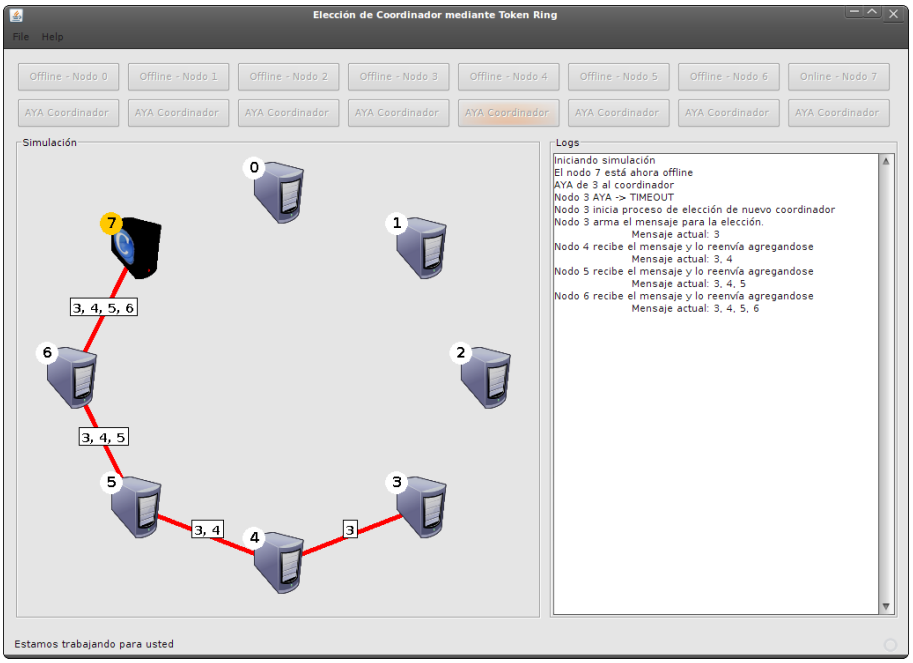

Figura 7: El nodo seis envia el mensaje al nodo siete.

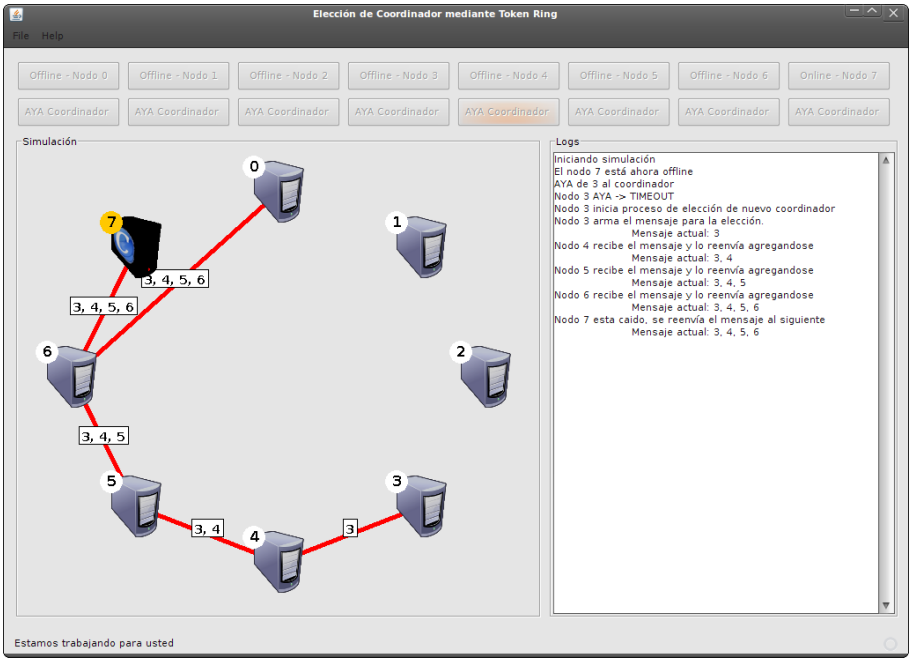

Figura 8: El nodo 6 envía el mensaje al nodo 7, como esta caido, él se lo reenvía al nodo 0

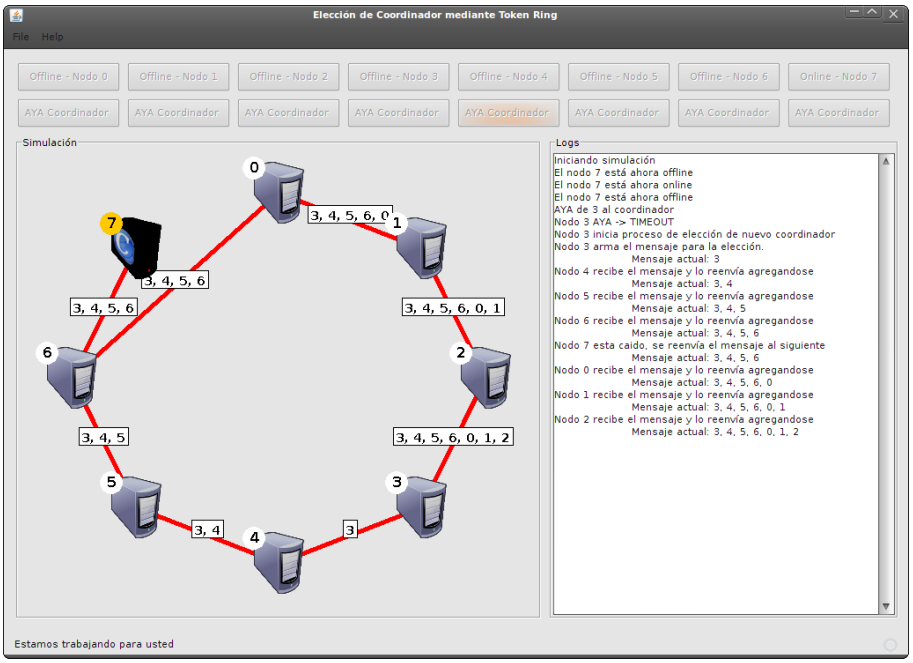

Figura 9: El mensaje termina de pasar por todos los nodos. El nodo tres, selecciona el mayor número en la lista.

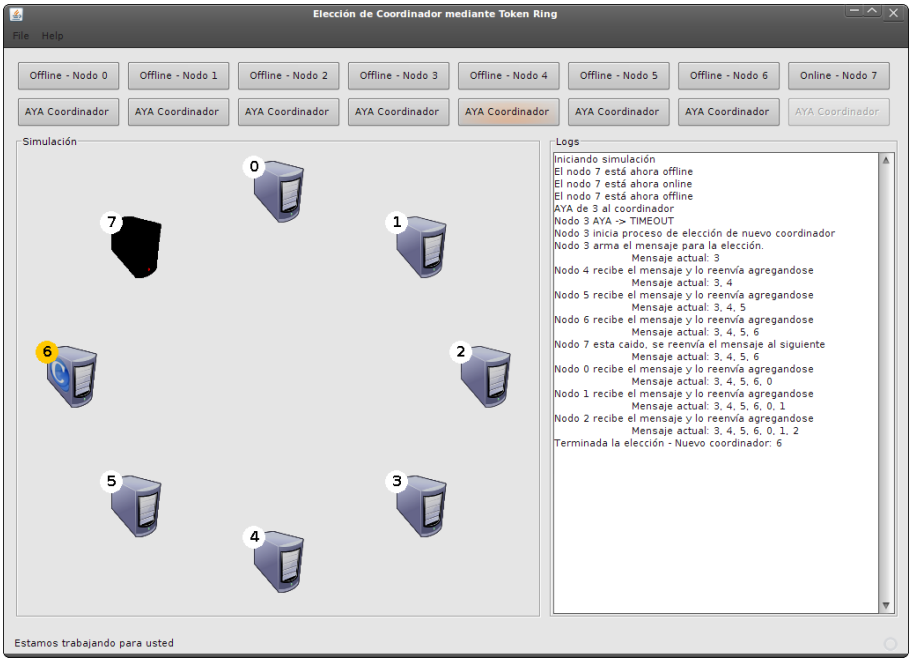

Figura 10: El nodo número tres avisa al resto que el nuevo coordinador es el número seis, este comienza a coordinar el sistema.

Otro ejemplo

Veamos un caso en el que más de un nodo no responde a los pedidos de los demás.

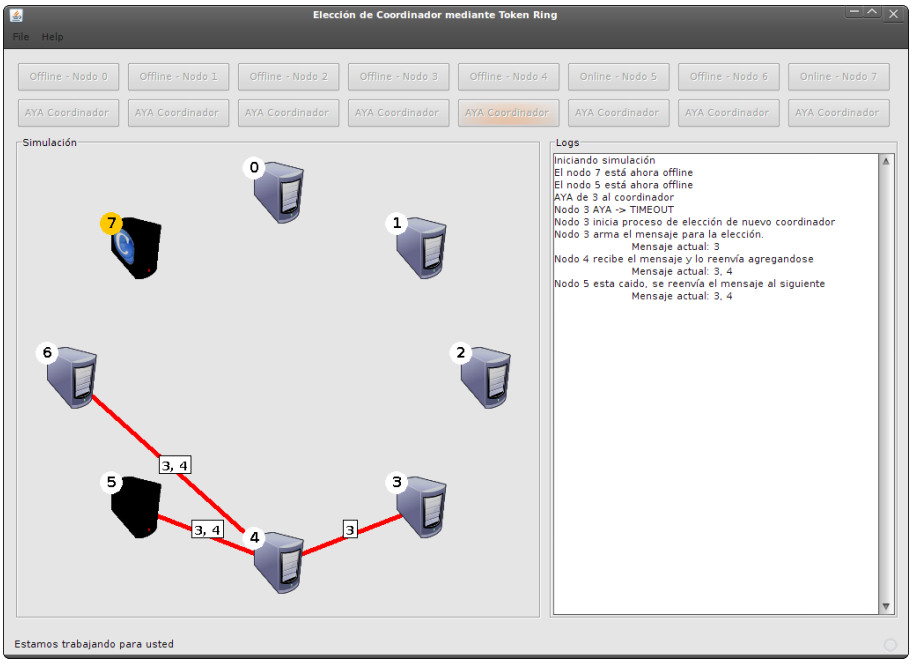

Figura 11: En este caso el nodo siete y el nodo cinco están caidos. Podemos observar como el nodo cuatro envía la lista al nodo seis al determinar que el cinco esta caído.

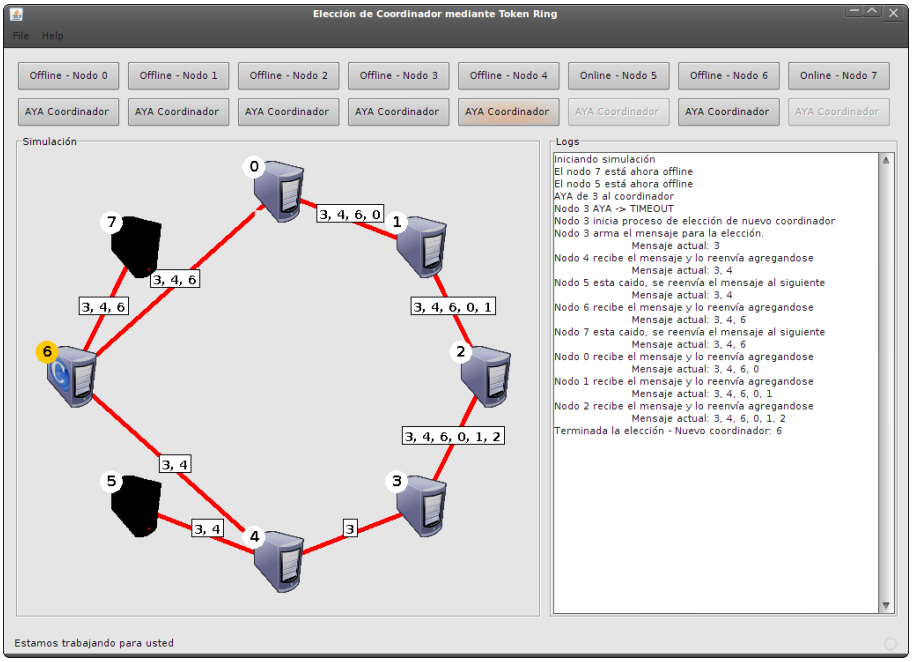

Figura 12: Terminada la elección del nuevo coordinador, podemos observar como el mensaje paso por todos los nodos activos y el nodo seis es el coordinador.

5. Semáforos

Un semáforo es una variable especial protegida que constituye el método clásico para restringir o permitir el acceso a recursos compartidos (por ejemplo, un recurso de almacenamiento del sistema o variables del código fuente) en un entorno de multiprogramacíon. Fueron inventados por Edsger Dijkstra y se usaron por primera vez en el sistema operativo THEOS. Con este simulador permitimos utilizar los semáforos de Dijkstra y simular su comportamiento al crear procesos que requieren el acceso a la zona crítica.

En la simulación el tiempo está dividido en turnos y a cada turno se ejecuta una etapa de un proceso que no se encuentre bloqueado.

La zona crítica es posible desocuparla automáticamente poniendo cuántos turnos quiere uno que un proceso esté hasta que sea obligado a liberarla o en modo manual para que un proceso pueda estar tanto tiempo como quiera.

5.1. Ejemplo de funcionamiento

Veamos un ejemplo para clarificar su comportamiento y forma de uso.

Figura 13: En la primer pantalla del simulador podemos elegir la cantidad de tipos de procesos y la cantidad de semáforos a utilizar, también podemos optar por cargar una instancia desde un archivo.

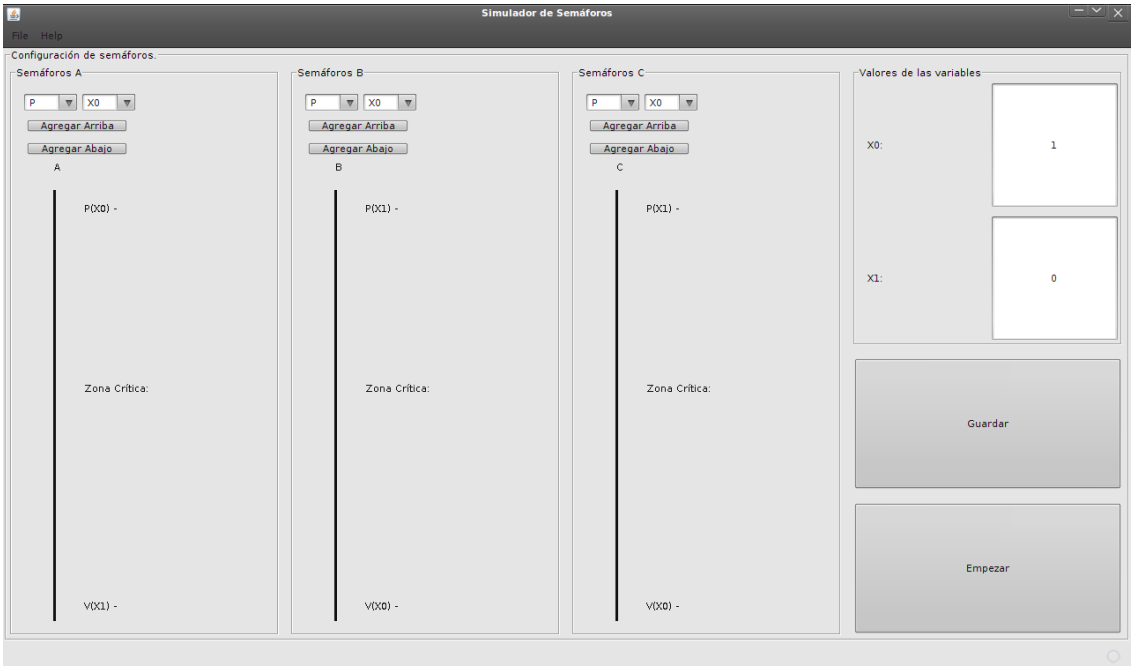

Figura 14: Ya en la segunda pantalla tenemos todas las herramientas para completar la configuración de nuestro sistema. Con los combos y los botones *Agregar Arriba*, *Agregar Abajo* y completando los valores de las variables podemos armar todos los semáforos que queramos modelar. En este ejemplo hemos completado con algunos operadores P y V, también *seteamos* los valores de *X0* y *X1*. Este ejemplo admite $A(B|C)$.

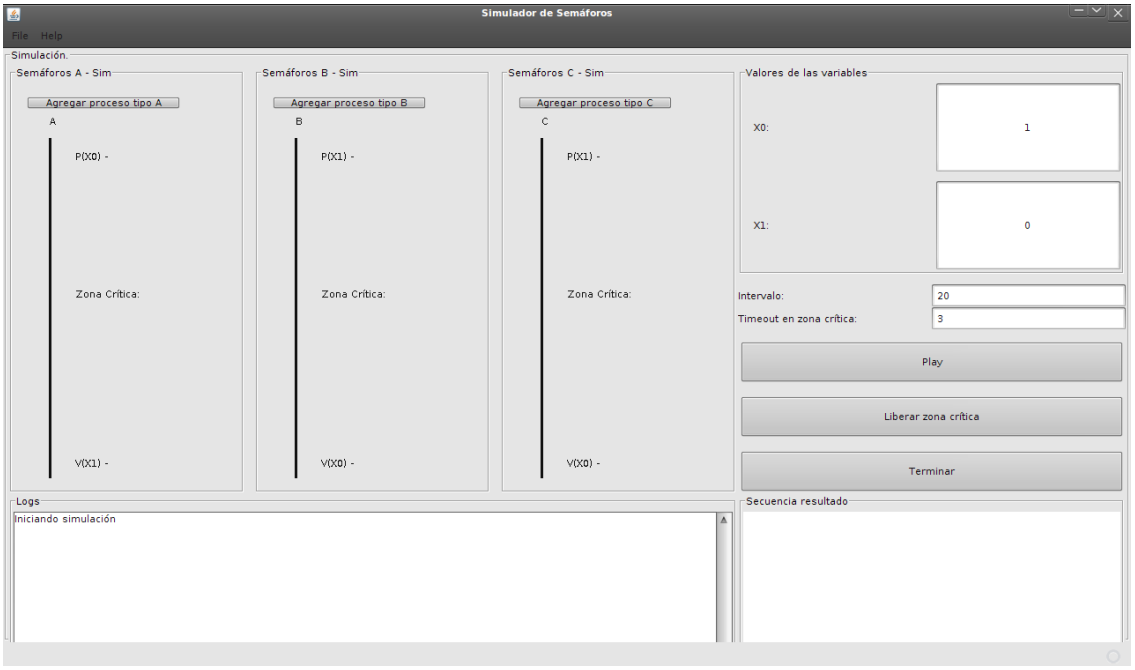

Figura 15: En la tercer pantalla podemos agregar procesos de cualquier tipo, *setear* los intervalos de tiempo correspondientes a la zona crítica y la velocidad de la simulación. El botón *Liberar zona crítica* saca al proceso que esté en la zona crítica en ese momento. Podemos además monitorear los valores de las variables de los semáforos, en el campo *Secuencia resultado* podemos ver los tipos de procesos que han utililzado la zona crítica y en el campo *Logs* podemos ver todo lo que ya ocurrió durante la ejecución.

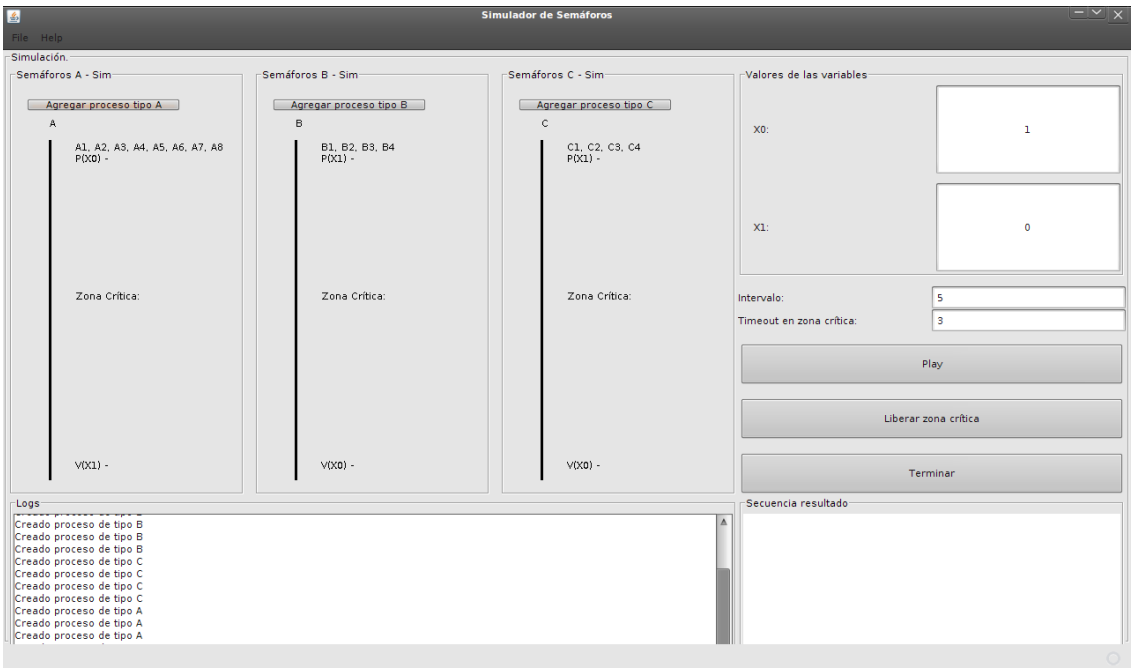

Figura 16: Agregamos distintos procesos de todos lo tipos para mostrar la ejecución del simulador.

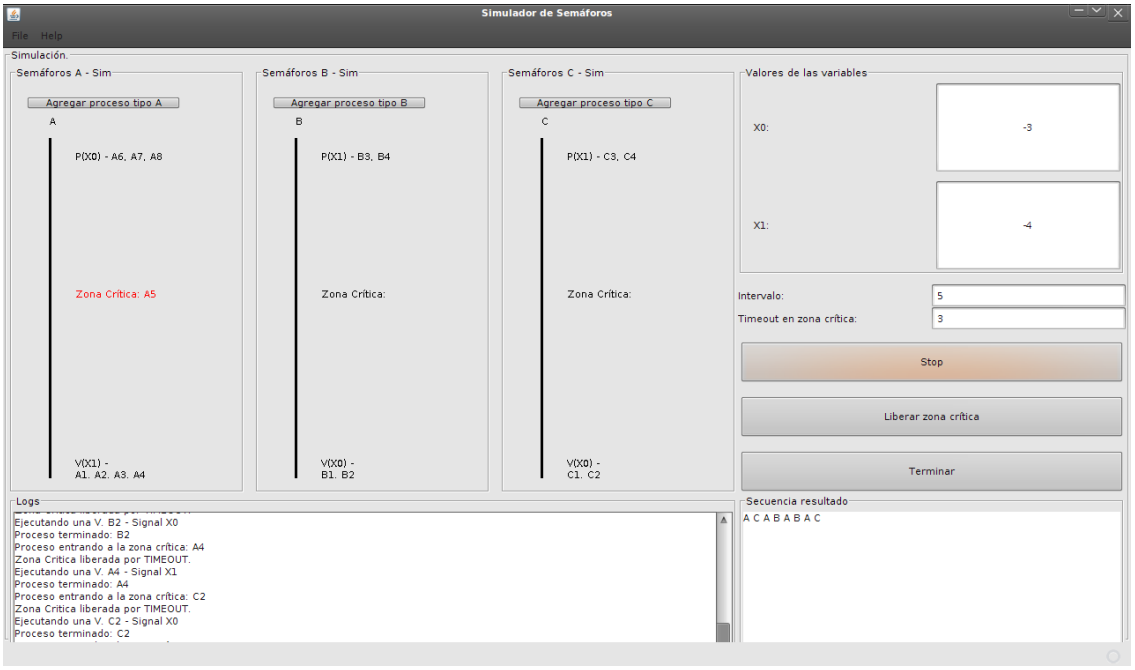

Figura 17: Por último, el simulador en acción. En este momento podemos notar al proceso *A5* ocupando la zona crítica. La *Secuencia Resultado* concuerda con lo esperado.

6. Manual de Usuario

Todos los simuladores se deben ejecutar desde la carpeta *ejecutables/NombreSimulador* ejecutando, por ejemplo, *java -jar Semaforo.jar*. Estos programas fueron desarrollados en plataforma Java 1.6, para poder ejecutarlos debe tener instalada la JRE 1.6 de Java, si no la tiene instalada puede descargarla desde http://www.java.com/en/download/manual.jsp . En la raíz del cd entregado encontraremos cuatro carpetas diferentes, en *codigo* se encuentra todo el código escrito durante el desarrollo del trabajo práctico, en *ejecutables* hay tres carpetas, una por simulador, en donde se encuentran los *.jar* ejecutables de cada simulador, en *Informe* se encuentra el informe del trabajo práctico, en *instanciasParaSimulacion* hay ejemplos de instancias que pueden ser cargadas desde los simuladores correspondientes, muchos de estos ejemplos fueron obtenidos de la práctica.