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	324/05	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 estn 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 est en un estado inseguro, aunque esto no implica que est bloqueado. As, 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 ejecucin no debe estar forzada por condiciones de sincronización.
- Debe haber un nmero fijo de recursos a utilizar y un nmero 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 se trata de un sistema que se encuentra en estado seguro o inseguro.

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

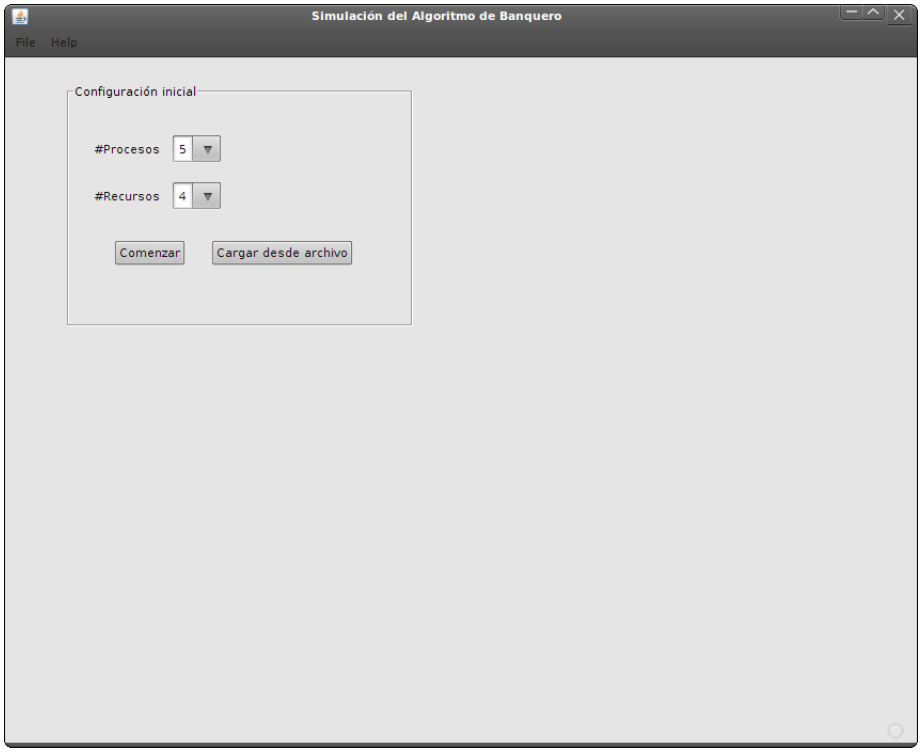

Figura 1: Esta es la primer pantalla del simulador, debemos escoger la cantidad de procesos y recursos o levantar una instancia guardada previamente desde el disco.

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

En este simulador intentamos mostrar cómo es llevada acabo la elección de un nuevo coordinador cuando el coordinador de una red en forma de anillo se cae por algún motivo. 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 suficiente para ilustrar el algoritmo sin pérdida de generalidad. Durante la simulación, cuando un nodo detecta que el coordinador actual dejó de responder mediante un timeout de un AYA (*Are you alive*), arma una lista que contendrá los nodos activos, se incluye a si mismo en esta lista y la envia a través de la red a la siguiente computadora, si esta computadora esta activa y responde con un IAA(*I am alive*), se agrega a la lista de los nodos activos, en caso contrario el nodo emisor le envia la lista de nodos activos a la siguiente computadora repitiéndose el proceso anterior. Una vez que la lista viaja por toda la red dando una vuelta entera al anillo vuelve al nodo emisor, este inspecciona la lista y selecciona el mayor de los números, este número es el que identifica al nuevo coordinador, el nodo que comenzo el proceso de seleccion avisa al resto de los nodos del nuevo coordinar elegido. Vale aclarar que si bien en este ejemplo el único nodo caido es el coordinador, podría pasar que otro nodo este caido también, en este caso cuando el mensaje se envie al nodo caido y no conteste entonces el mensaje será enviado nuevamente al siguiente nodo, de la misma forma que se hace cuando el mensaje es enviado al coordinador cuando este esta caido. Además si un nodo envia un AYA en un instante en el que el coordinador esta activo, este contestará inmediatamente y no será necesario ningún algoritmo de selección.

Veamos ahora un ejemplo de ejecución del simulador: Supongamos que tenemos una red de ocho computadoras numeradas del cero al siete. El nodo siete es el coordinador, pero por alguna razón deja de responder y el nodo número tres realiza un AYA al nodo coordinador para saber si esta activo, al no responder luego de un lapso de tiempo y dar *timeout* el nodo número tres arma la lista de nodos activos, se incluye en esta y la envia al siguiente nodo, en este caso será el numero 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

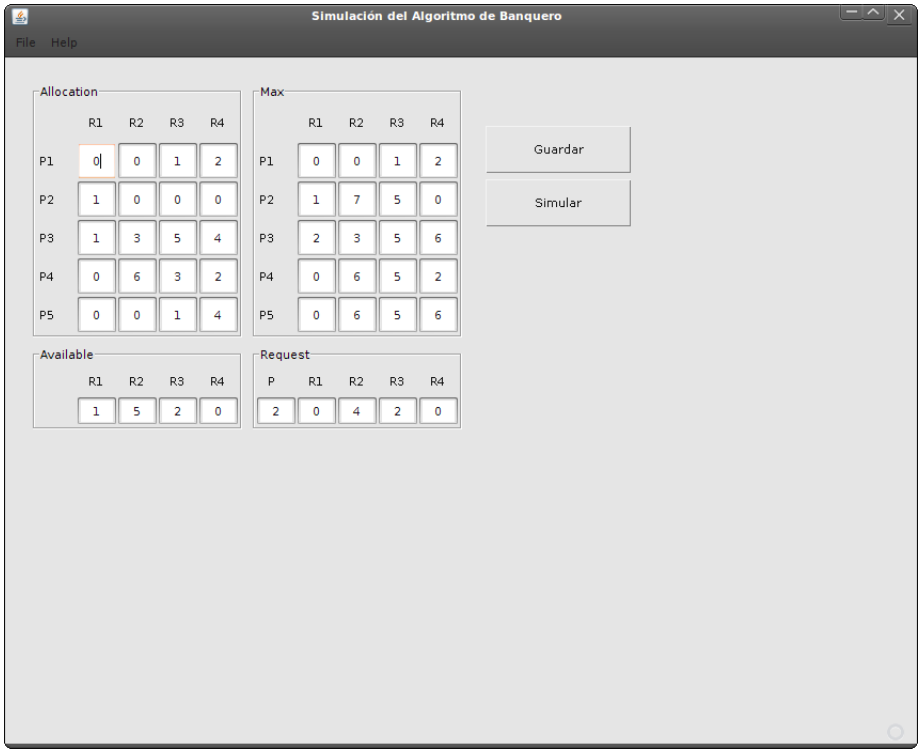

Figura 2: Aquí ya vemos la siguiente pantalla, en la que debemos ingresar los valores de entrada de la simulación. *Allocation* es la matriz que refleja los recursos que tiene alocado cada proceso, *Max* es el máximo de recursos que un proceso puede usar, *Available* es la cantidad de recursos de cada tipo que no están asigandos, *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*.

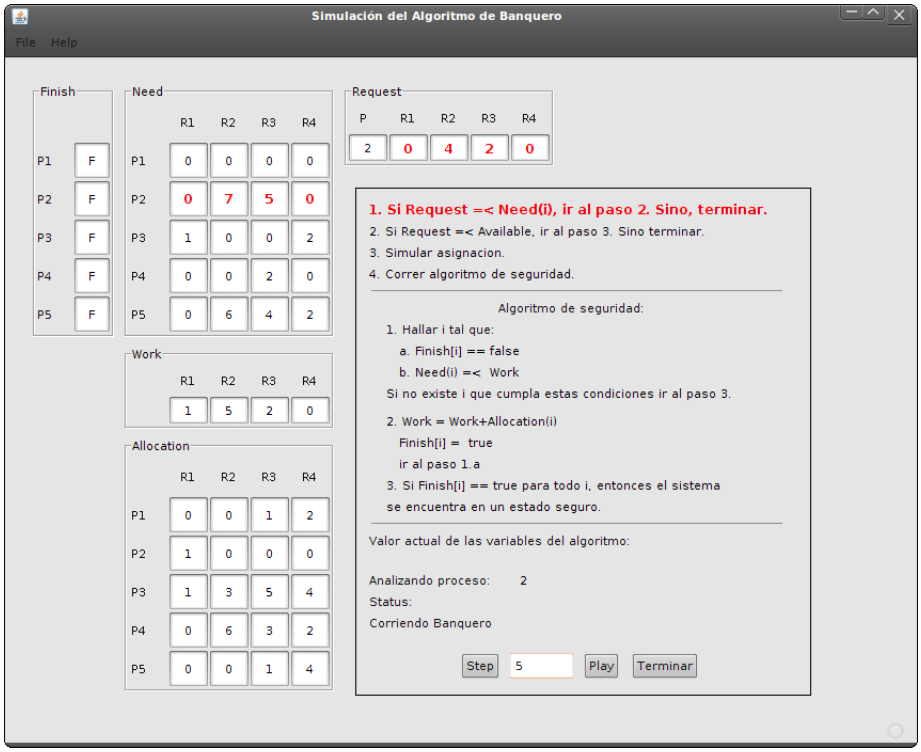

Figura 3: Esta es la última pantalla del simulador, en ella vemos los valores de las distintas matrices que participan en el algoritmo. El vector *Finish* nos indicará cuales de los procesos pueden terminar, si todo el vector *Finish* queda en *True*, querrá decir que el sistema esta en un estado seguro. El botón *Step* avanza el algoritmo un paso, podemos presionarlo repetidas veces hasta finalizar la simulación, también podemos elegir un intervalo 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.

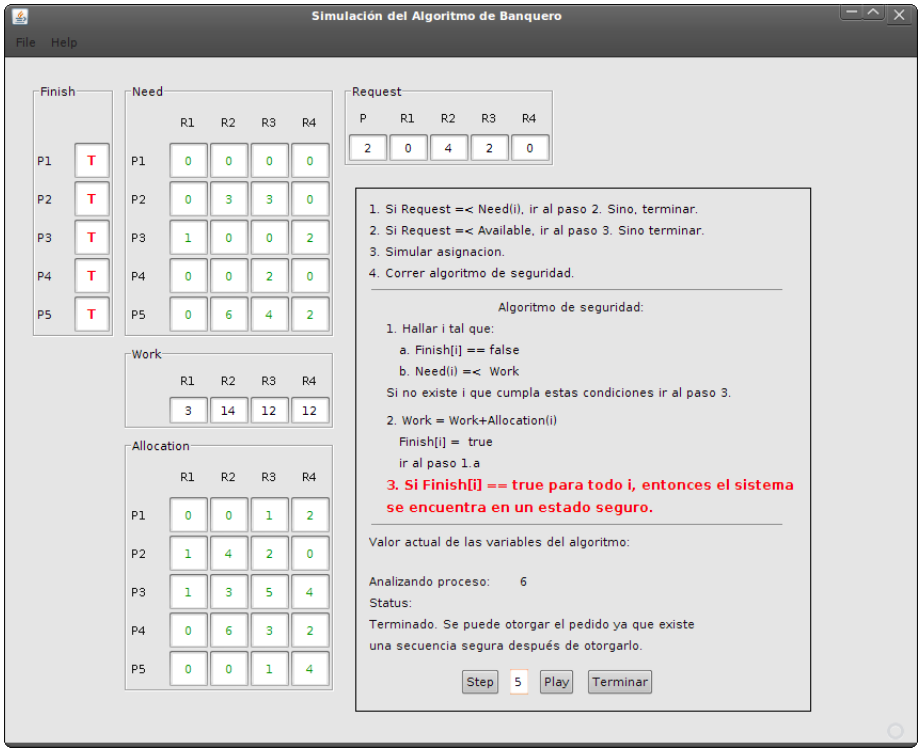

Figura 4: Aqui ya ha terminado la simulación, podemos ver el vector de *Finish* completamente seteado en *True*, eso nos indica que existe una secuencia segura en este sistema.

llegará nuevamente al número tres, que al verse incluido en la lista de nodos activos sabrá que debe realizar la selección del nuevo coordinador. Para esto toma el mayor valor de la lista, resultando este el número seis y le envia un mensaje al nodo número seis para que comience a coordinar a la red.

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

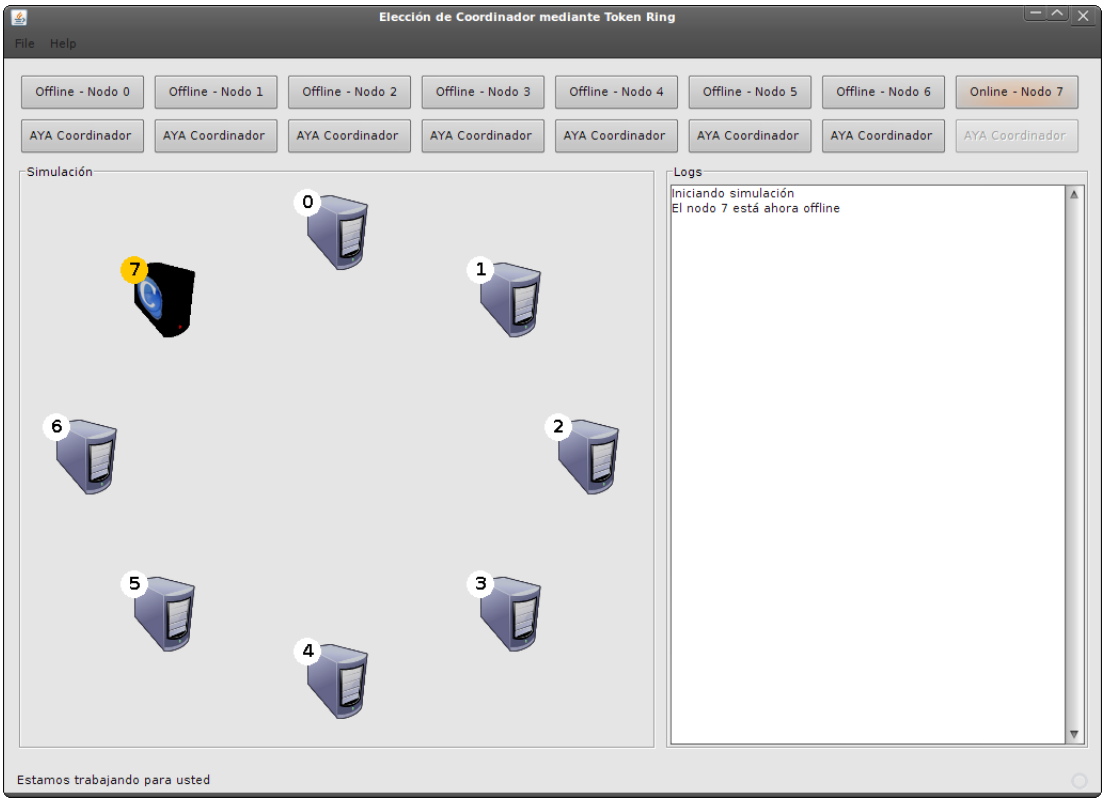

Figura 5: Aquí vemos al simulador con el nodo siete caido.

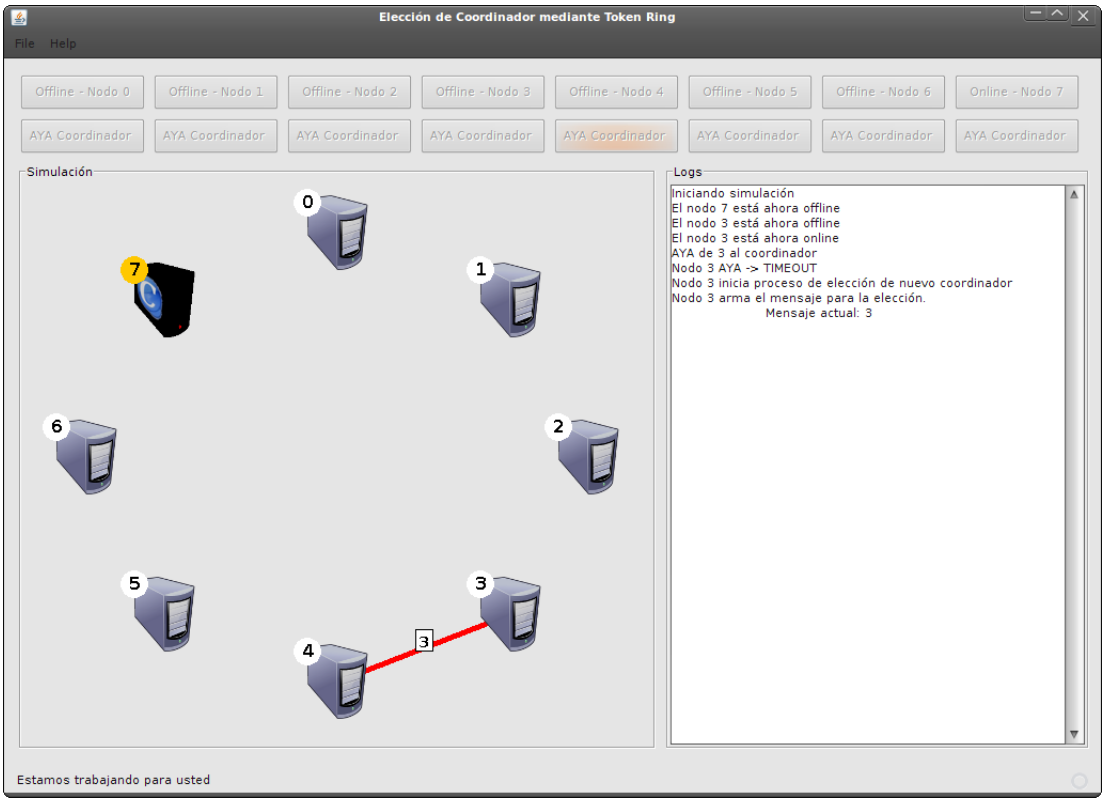

Figura 6: El nodo tres no recibe respuesta del coordinador y comienza el proceso de elección.

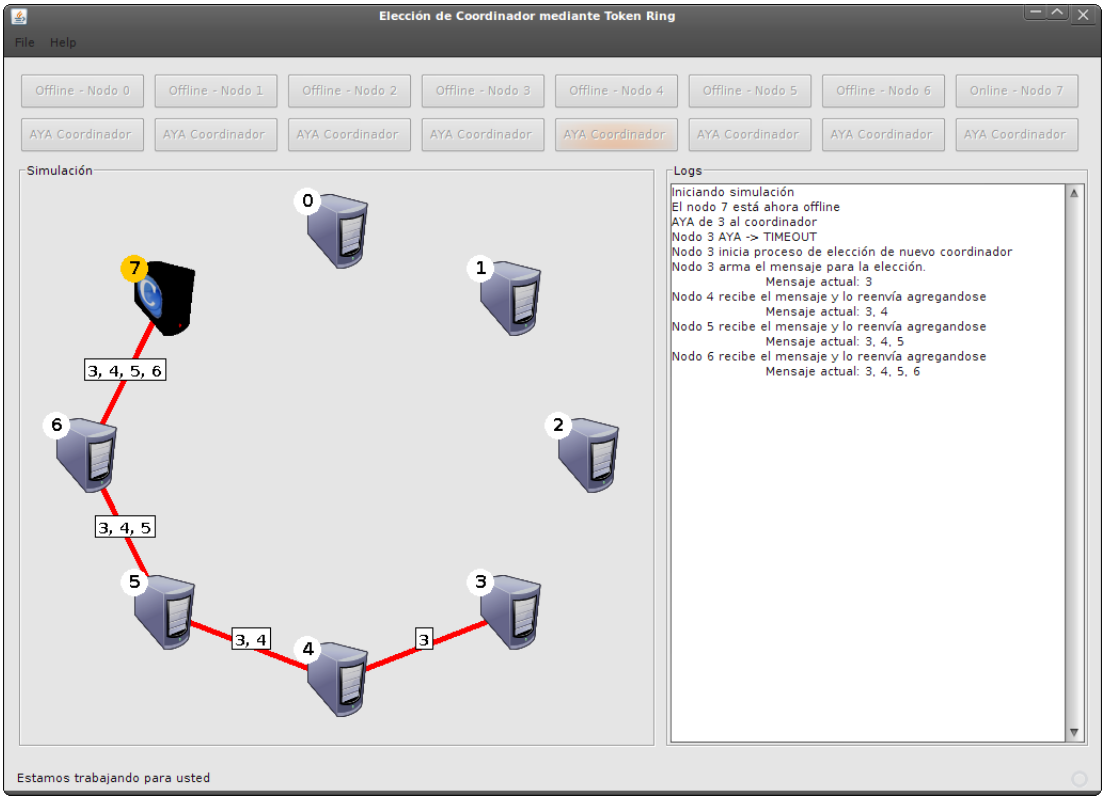

Figura 7: El nodo seis envia un AYA al nodo siete.

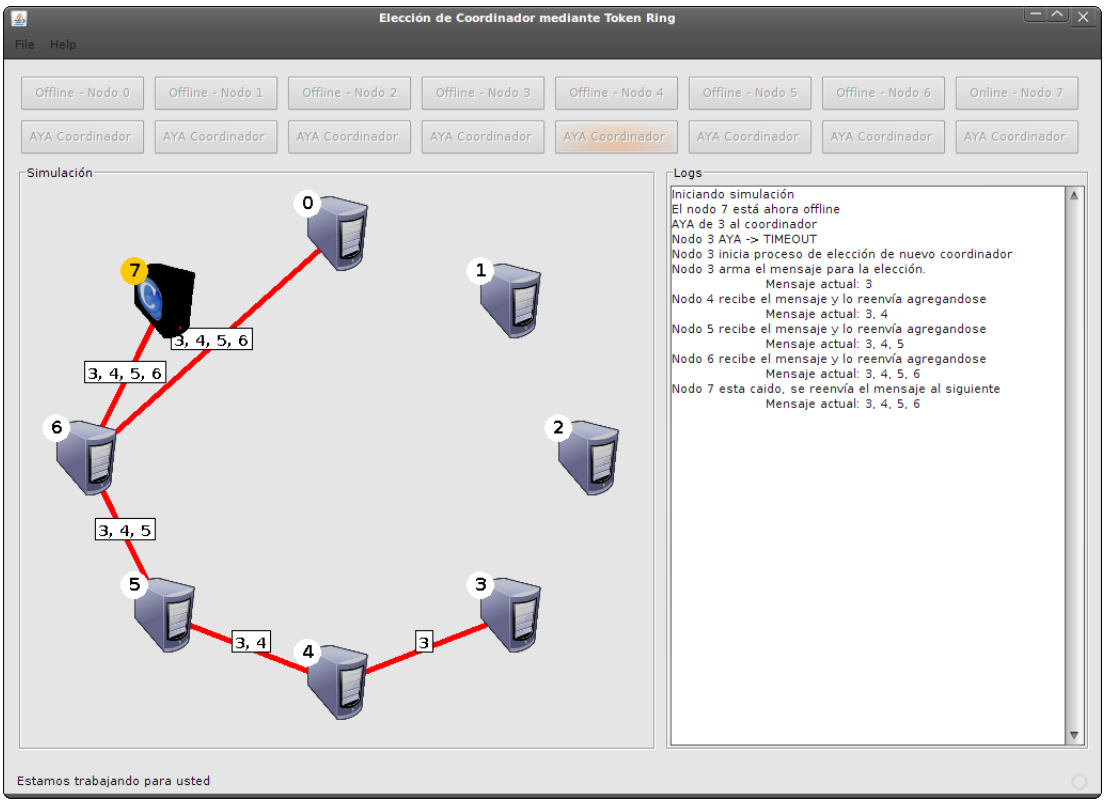

Figura 8: Después de un timeout el nodo seis envia un mensaje de AYA al nodo cero, como este contesta entonces le envia la lista con todos lo nodos activos hasta el momento.

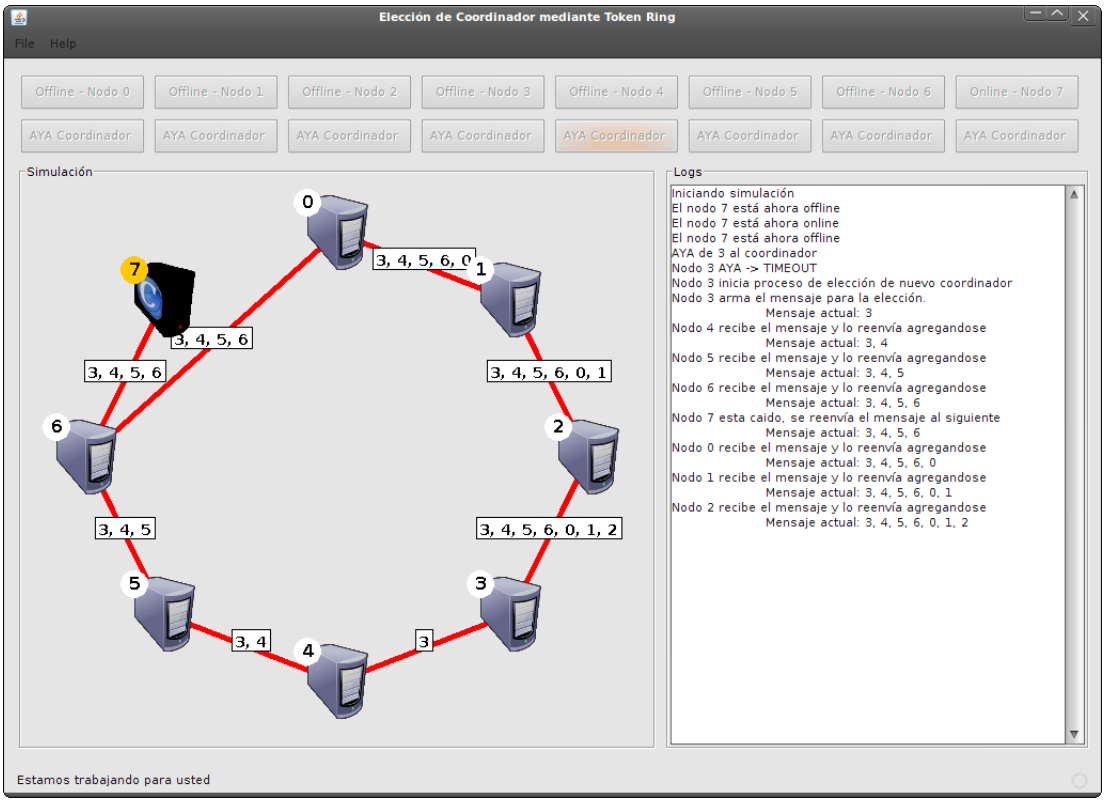

Figura 9: El mensaje termina de pasar por todos los nodos y el nodo tres, selecciona el mayor de los números en la lista.

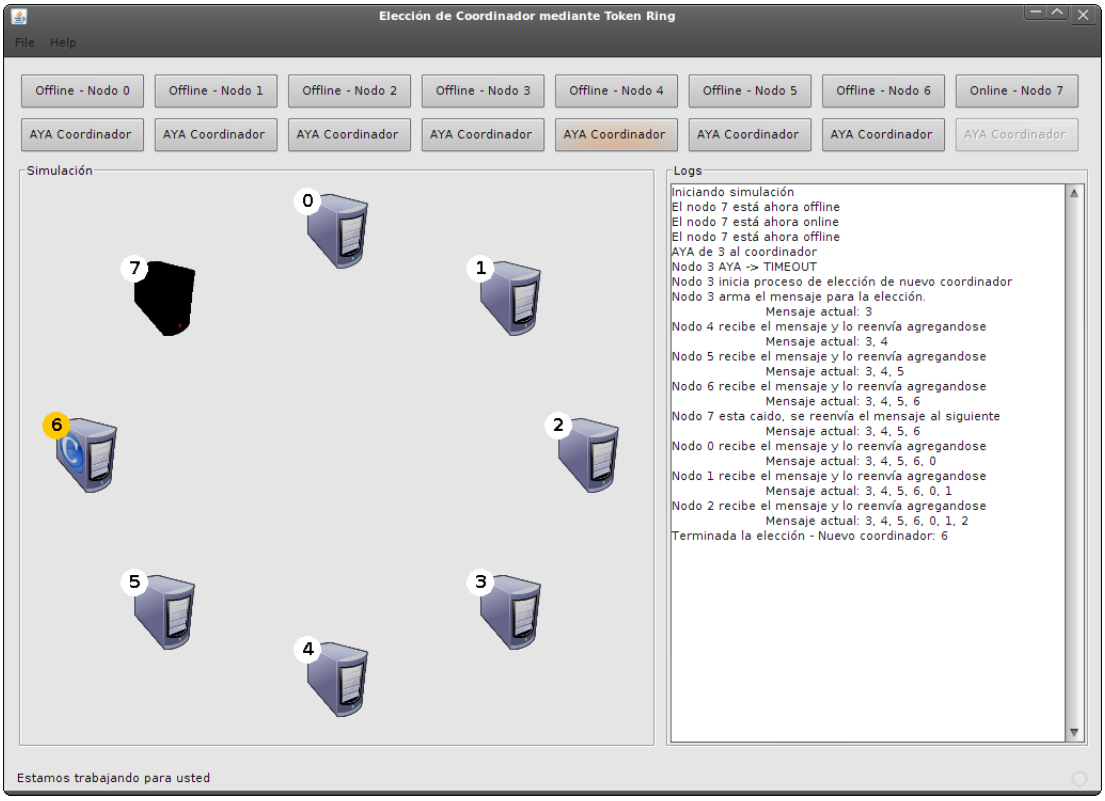

Figura 10: El nodo número tres avisa al resto que el nuevo coordinador es el número seis, este comienza a coordinar al todo el sistema.

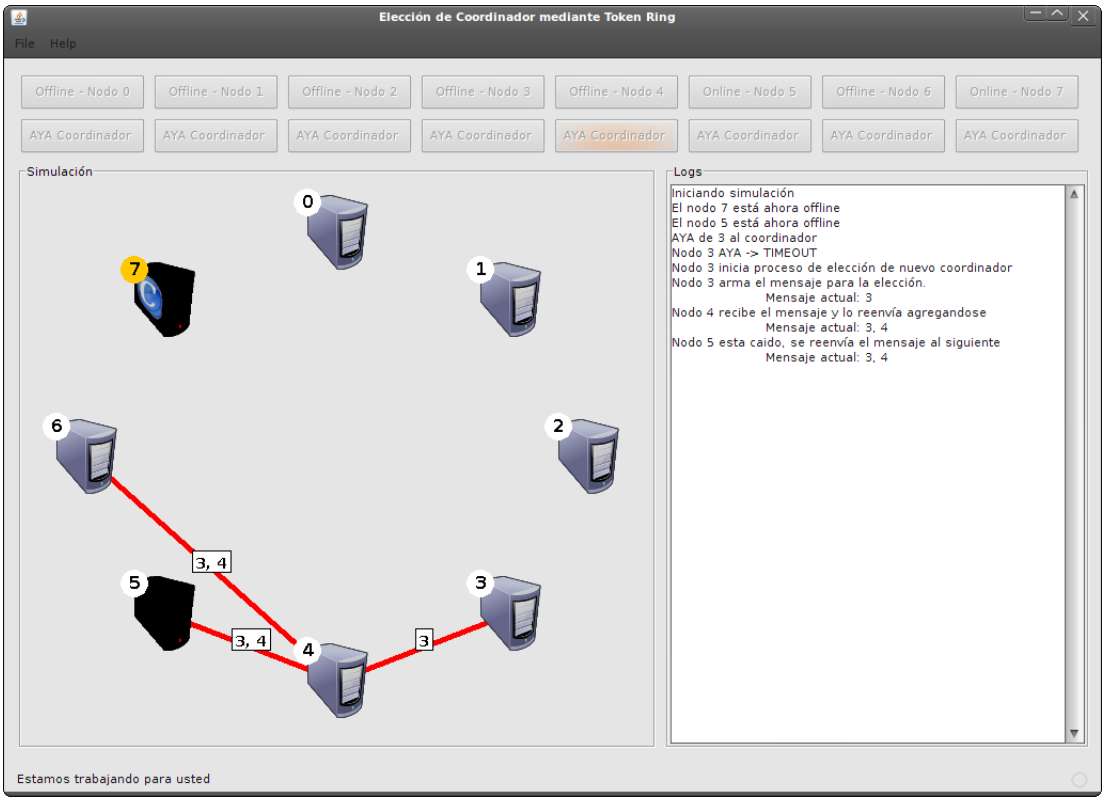

Figura 11: En este caso el nodo siete y el nodo cinco estan caidos, podemos observar como el nodo cuatro envia la lista al nodo seis al determinar que el cinco esta caido.

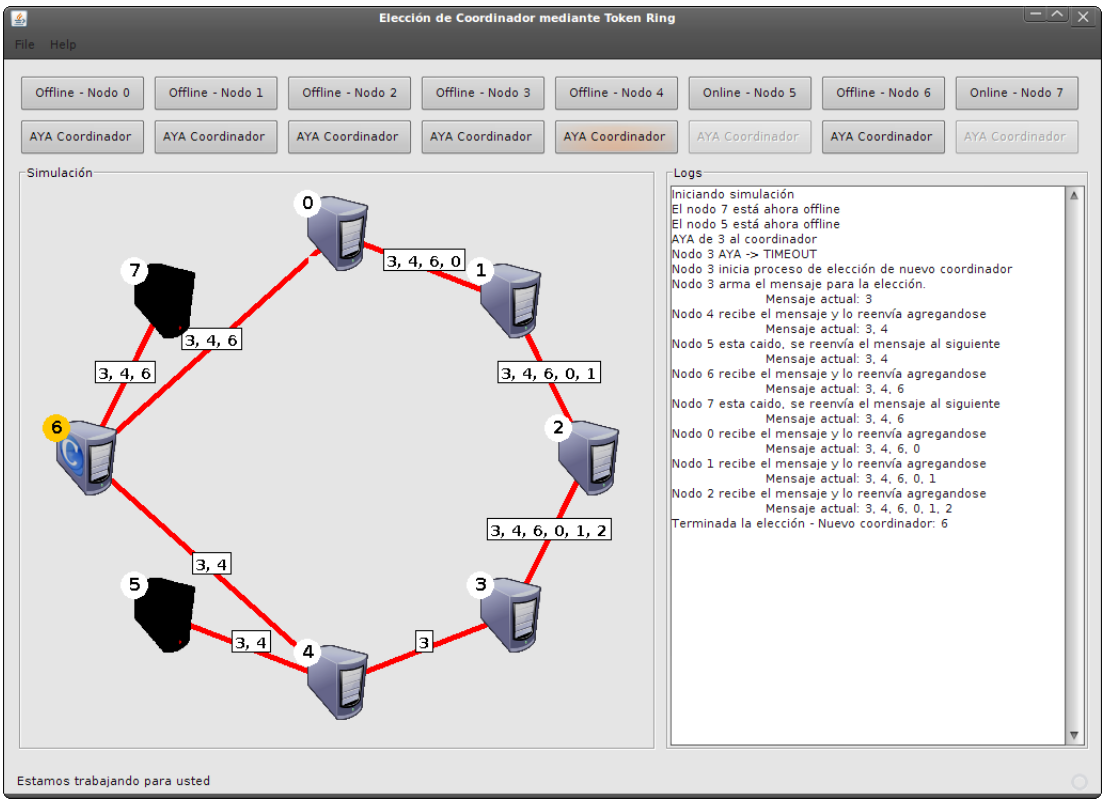

Figura 12: Terminada la elección del nuevo coordinador, podemos observar como el mensaje paso por todos los nodos activos y el nodo seis es el coordinador.

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 los datos desde un archivo.

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 cdigo 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 armar los semáforos de Dijkstra y simular su comportamiento al crear procesos que requieren el acceso a la zona crítica. Veamos un ejemplo para clarificar su comportamiento y forma de uso.

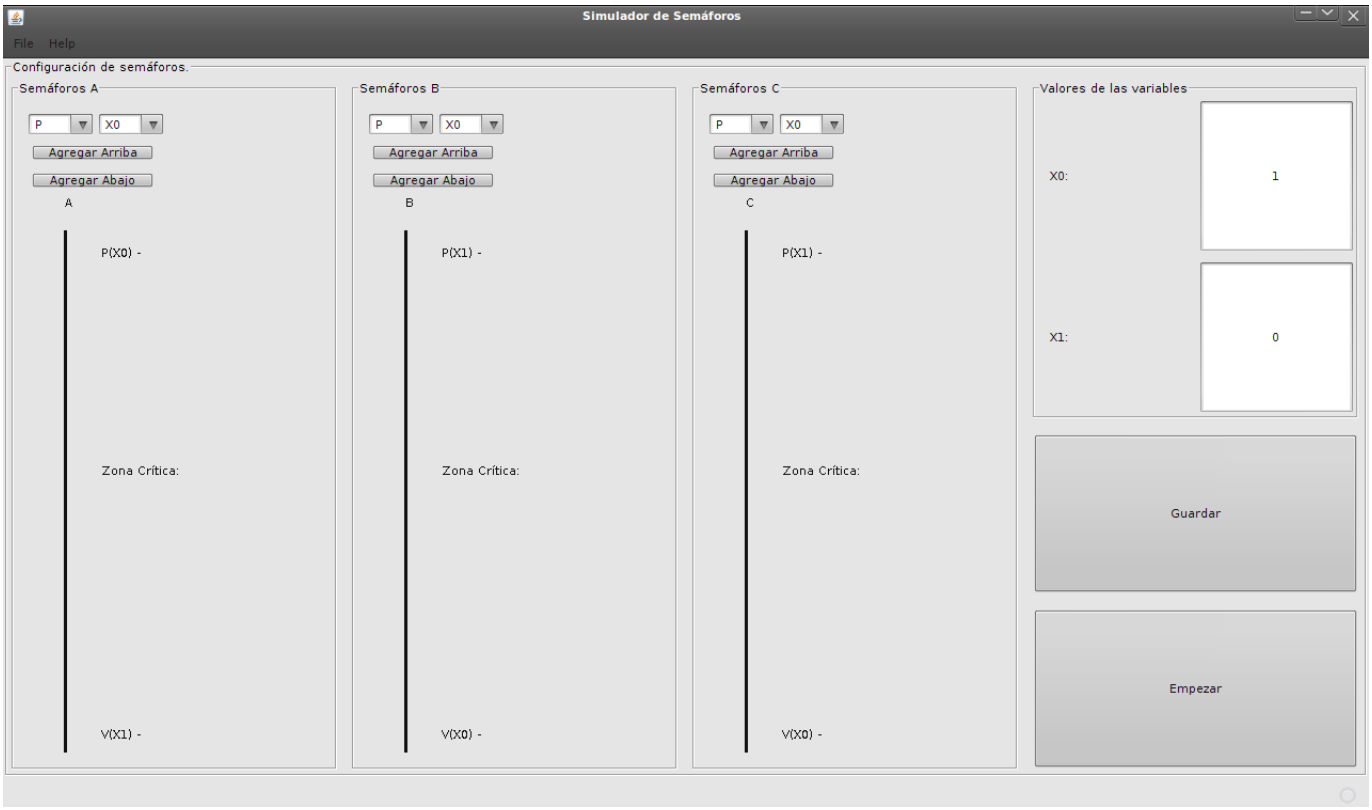

Figura 14: Ya en la segunda pantalla tenemos todas las herramientas para completar la configuración de nuestro semáforo. Con los combos y los botones *Agregar Arriba*, *Agregar Abajo* y completando los valores de las variables podemos armar todos los semáforos que queramos modelar. En este ejemplo hemos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

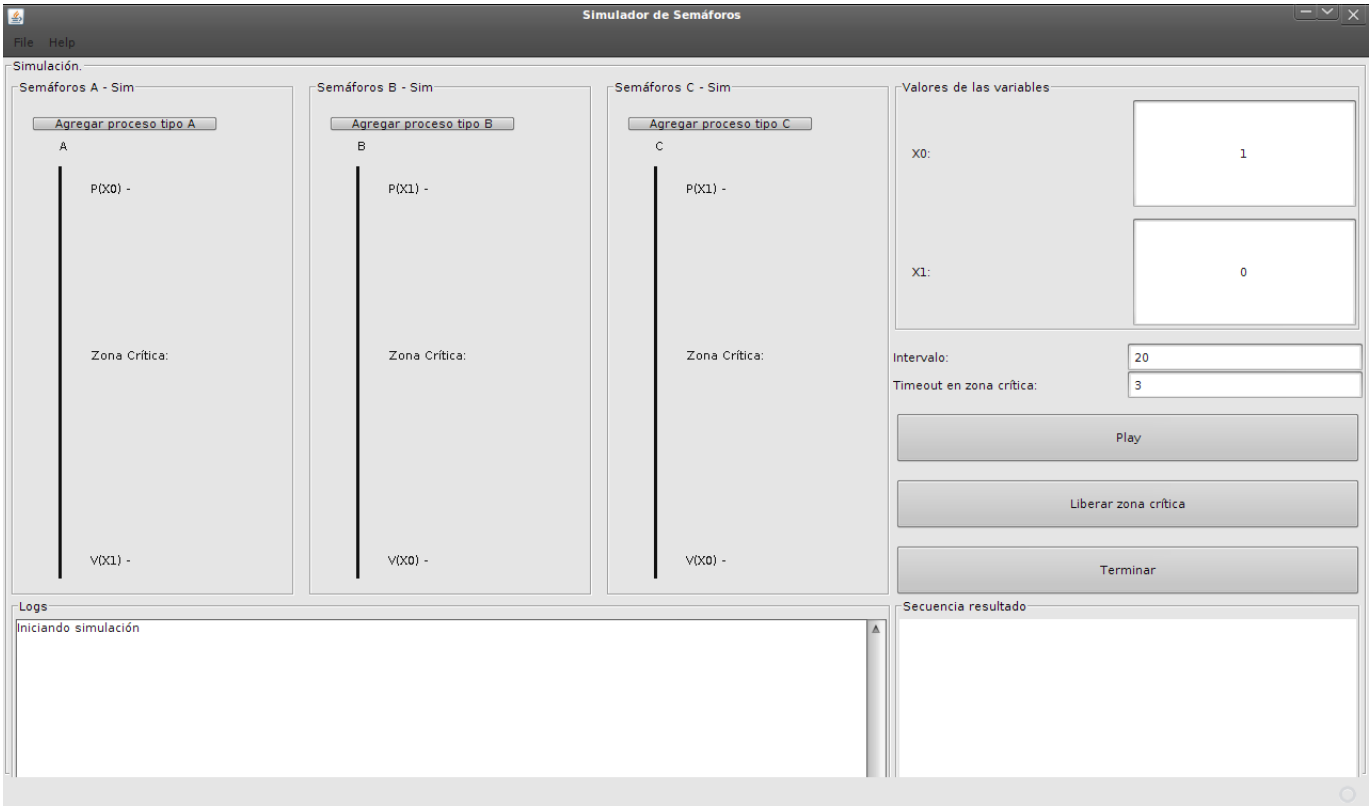

Figura 15: En la tercer pantalla podemos agregar procesos de cualquier tipo, setear los intervalos de tiempo correspondientes a la zona crítica y la velocidad del algoritmo en ejcutarse. El botón *Liberar zona crítica* saca al proceso que esté en la zona crítica en ese momento dando lugar al siguiente proceso. 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 en cada ejecución.

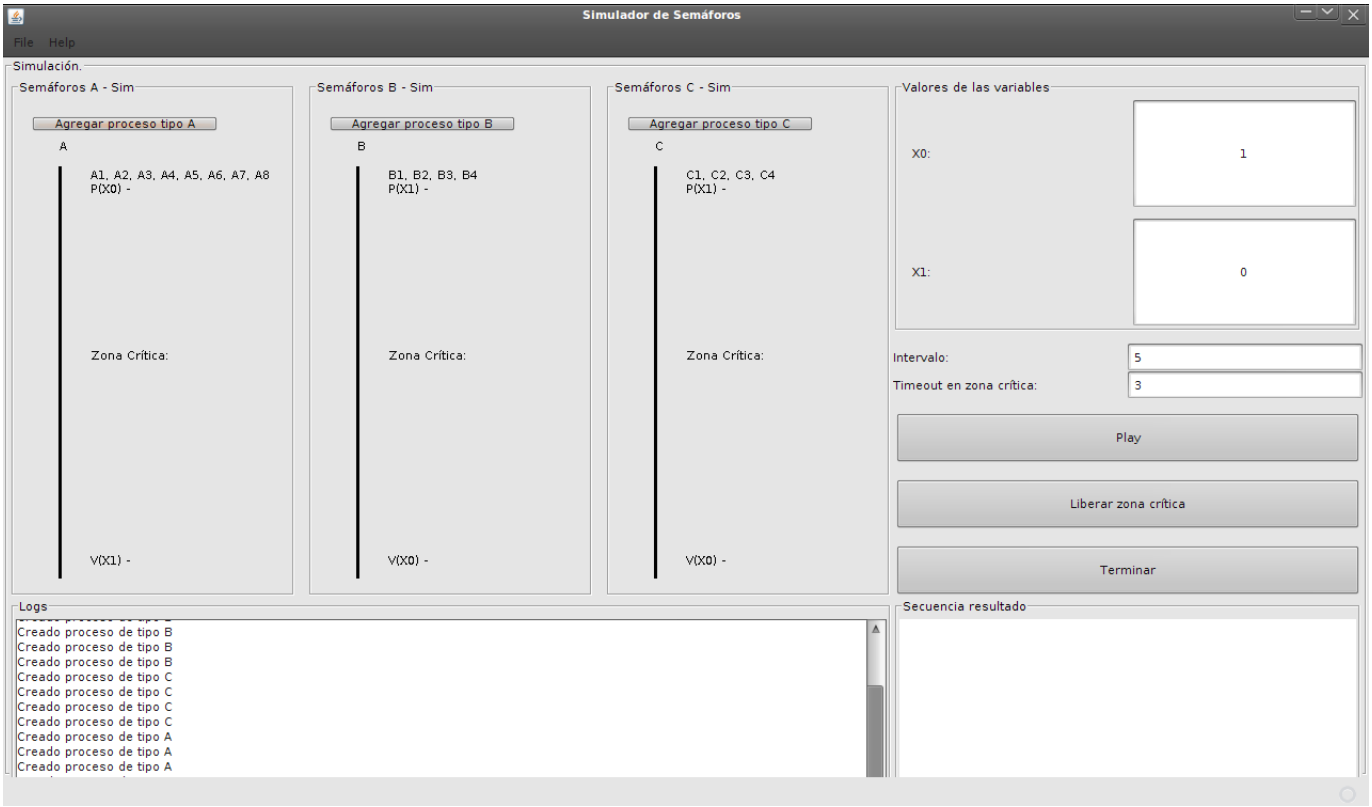

Figura 16: Agregamos distintos procesos de todos lo tipos para mostrar la ejecución del simulador.

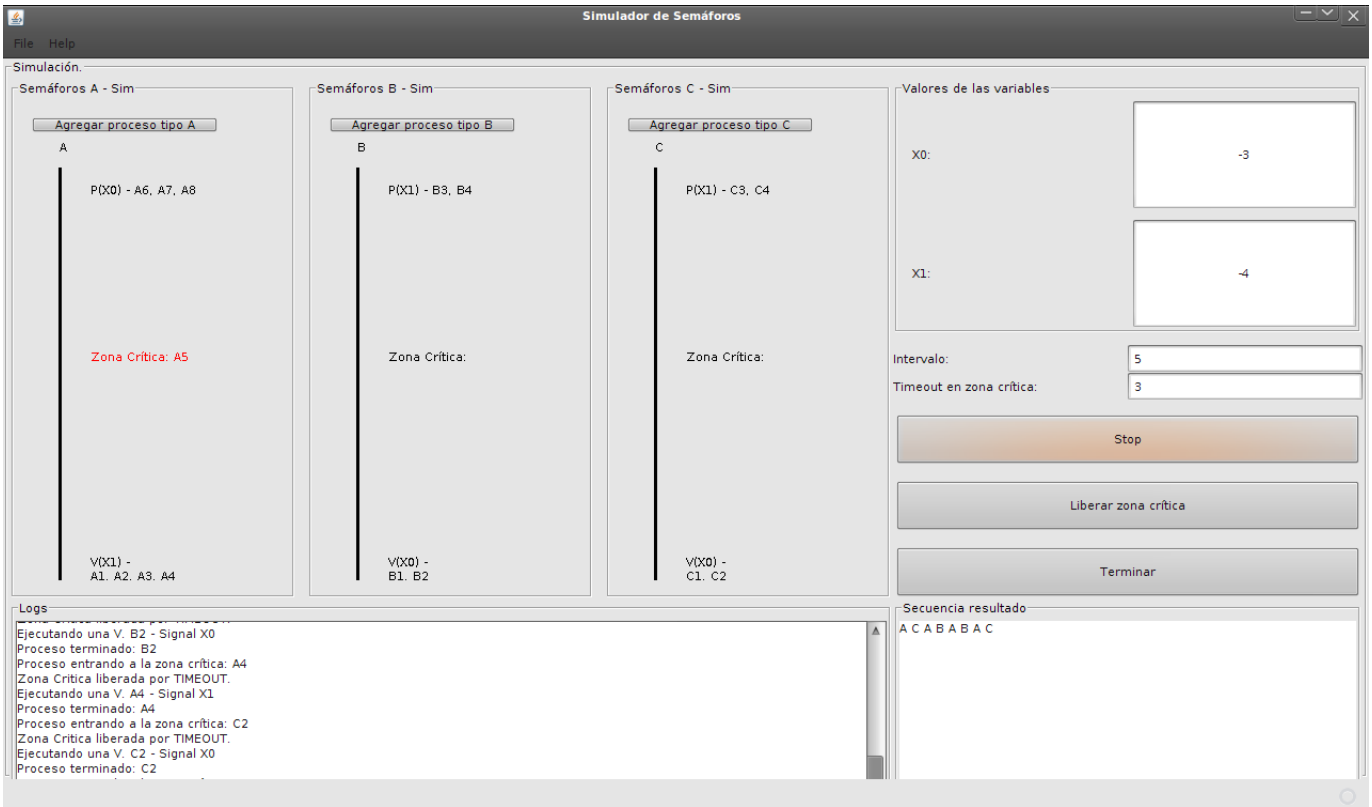

Figura 17: Por último, el simulador en acción. En este momento podemos notar al proceso *A5* ocupando la zona crítica. La *Secuencia Resultado* concuedra con lo esperado.

6. Manual de Usuario

Todos los simuladores se deben ejecutar desde la carpeta *dist* 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 .