

Universidad Nacional del Nordeste
Facultad de Ciencias Exactas y Naturales y
Agrimensura

Trabajo Final de Aplicación:
Entorno Web para la Enseñanza-Aprendizaje de Algoritmos de
Sistemas Operativos



Alumno Nelson Fabian Rodriguez - LU: 32.945
Prof. Director: Dr. David Luis La Red Martinez
Licenciatura en Sistemas de Información
Corrientes-Argentina
2011

A mis padres, por confiar en mí y otorgarme su apoyo incondicional.

A mi hermano, un camarada excepcional.

A mis hermanas, por su compañía.

A mis abuelos, tíos y primos, por formar parte de mi vida.

A mis amigos, los que siempre estuvieron a mi lado.

A mis profesores, por todo lo brindado.

Prefacio

Este Trabajo Final de Aplicación fue realizado teniendo como objetivo principal que los alumnos de la cátedra de Sistemas Operativos, de la Facultad de Ciencias Exactas y Naturales y Agrimensura, perteneciente a la Universidad Nacional del Nordeste, puedan ejecutar algoritmos de administración de recursos de los sistemas operativos en un entorno web. Estos algoritmos corresponden a los distintos casos de estudio analizados en el laboratorio de la cátedra.

Para lograr esto se desarrollo una serie de applets en el lenguaje Java que serán ejecutados desde un sitio web, en donde se podrán observar el funcionamiento de algunos algoritmos de administración de recursos de los sistemas operativos convencionales y sistemas operativos ditribuidos, con solo acceder a un ordenador con conexión a Internet.

Este trabajo presenta en cada caso de estudio un marco teórico explicando el funcionamiento de los algoritmos desarrollados y analizados.

En el **Capítulo 1**, se desarrolló una breve introducción del trabajo realizado, donde se describen los objetivos propuestos, las herramientas utilizadas y los algoritmos de los casos de uso tratados.

En el **Capítulo 2**, se analizaron los Algoritmos de Planificación del Procesador, donde se podrán comprender el funcionamiento y los distintos estados de los procesos internos de un sistema operativo convencional.

Se observaron distintas estrategias de planificación que utilizan los sistemas operativos de este tipo, como ser, FIFO, Round Robin, HRN y RNM, cada una de estas explicadas con detalle.

Se llevó a cabo la simulación por medio del desarrollo de un applet, donde se debe introducir la cantidad de procesos, la cantidad de ciclos de control y las estrategias que se desea simular, de esta manera se podrá observar un informe detallado de cada una de las estrategias seleccionadas al inicio de la simulación.

En el **Capítulo 3**, se analizaron las estrategias de reposición de páginas en la memoria principal determinado por el esquema FIFO (Primero en Entrar Primero en Salir), se detallan algunas de sus características y se estudia lo que se llama Anomalía Belady o Anomalía FIFO.

Se estudia la llamada Anomalía Belady o Anomalía FIFO por medio del desarrollo de un applet, donde se deberá introducir la cantidad de marcos de página para iniciar la simulación.

En el applet desarrollado se podrá observar el resultado de la simulación

por medio de reportes escritos y gráficos, teniendo en cuenta distintas cargas de trabajo.

En el **Capítulo 4**, se llevó a cabo el Análisis del Rendimiento de un Subsistema de Disco de Una Petición a la Vez, donde se observó el rendimiento de un disco con un solo brazo cuando es sometido a distintas cargas de trabajo.

Se simuló a un conjuntos de peticiones de operaciones de acceso a los discos, que configuran cantidades de trabajo distintas a ser atendidas, calculando los tiempos de llegadas, tiempo de servicio, tiempo de espera, capacidad de la cola, etc.

Para llevar a cabo esta simulación se desarrolló un applet, donde se deben ingresar los valores máximos de las variables μ (mu); λ (lambda) y el número de peticiones pendientes para iniciar la simulación.

En el applet se pueden observar informes detallados de los cálculos efectuados, un análisis estadístico de los resultados y un gráfico donde se muestran los datos obtenidos.

En el **Capítulo 5**, se llevó a cabo el Análisis del Rendimiento de un Subsistema de Disco de Varias Peticiones a la Vez, que a diferencia del capítulo anterior, en este caso se observa el rendimiento de un disco con varios brazos independientes entre sí para atender solicitudes.

Esto requiere que el subsistema de disco disponga de un número de brazos de acceso, que además sean independientes en sus movimientos, de tal manera que cada uno de ellos pueda atender a una petición cuando es sometido a distintas cargas de trabajo.

En este caso de estudio también se realiza la simulación calculando los tiempos de llegadas, tiempo de servicio, tiempo de espera, capacidad de la cola, etc.

Para llevar a cabo esta simulación se desarrolló un applet donde se deben ingresar los valores máximos de las variables μ (mu); λ (lambda) y el número de peticiones pendientes.

En el applet, se podrán observar informes detallados de los cálculos efectuados, un análisis estadístico de los resultados y un gráfico donde se muestran los datos obtenidos.

En el **Capítulo 6**, se analizó la ejecución concurrente de los hilos y la sincronización de los mismos en los sistemas operativos convencionales.

Se llevó a cabo la simulación de este caso de estudio por medio del desarrollo de un applet que implementó el problema de procesos productores y consumidores, donde los procesos productores generan información en un buffer y los procesos consumidores retiran información del mismo buffer.

El applet se diseñó de manera tal que permitiera generar en un arreglo el seguimiento de dicha simulación para posteriormente efectuar el análisis de la forma en que se ha desarrollado la misma.

En el **Capítulo 7**, se analizó un algoritmo de sincronización de procesos en sistemas operativos distribuidos para el uso de recursos críticos, el caso de estudio analizado es el llamado, Exclusión Mutua.

En este caso se planteó un esquema de control centralizado similar al de los sistemas operativos convencionales, donde existe un proceso coordinador para un conjunto determinado de procesos.

El proceso coordinador actúa como administrador, otorgando el recurso crítico en el caso de que esté libre, bloqueando en el caso de que el recurso crítico esté siendo usado por otro proceso y desbloqueando los recursos críticos en el caso de que algún proceso termine de forma irregular cuando tenga apropiado algún recurso crítico.

Se llevó a cabo la simulación de este caso de estudio por medio del desarrollo de un applet donde, se hizo uso de hilos para representar a los procesos y se hizo uso de arreglos para representar cada región crítica.

Para dar inicio a la simulación de este caso de estudio, se debe introducir en el applet, la cantidad de procesos, la cantidad de regiones críticas y el tiempo de simulación en segundos.

En el **Capítulo 8**, se describe cada componente de la aplicación web desarrollada para contener a los applets que simulan cada uno de los algoritmos de administración de recursos de sistemas operativos desarrollados a lo largo de este trabajo.

Corrientes, Noviembre de 2011.

Índice general

1. Introducción	1
2. Algoritmos de Planific. del Procesador	3
2.1. Introducción	3
2.2. Objetivo del Caso de Estudio	3
2.3. Descripción del Problema Planteado	4
2.4. Estados de Procesos	5
2.5. Descripción de los Algoritmos Utilizados	8
2.6. Applet Desarrollado	9
2.7. Datos y Ejecuciones	22
2.8. Resultados y Conclusiones	22
3. Anomía Belady	24
3.1. Introducción	24
3.2. Objetivo	24
3.3. Reposición de Página por el Sistema de Primero en Entrar – Primero en Salir (FIFO)	25
3.3.1. Conceptos Generales de Gestión de Memoria	25
3.3.2. Reposición de Página por el Sistema FIFO	27
3.4. Applet Desarrollado	27
3.5. Datos y Ejecuciones	44
3.6. Resultados y Conclusiones	45
4. Subsistema de Una Petición a la Vez	46
4.1. Introducción	46
4.2. Objetivo del Caso de Estudio	46
4.3. Descripción del Problema Planteado	47
4.4. Descripción del Algoritmo Desarrollado	54
4.5. Solución Propuesta	54

4.6. Applet Desarrollado	56
4.7. Datos y Ejecuciones	74
4.8. Resultados y Conclusiones	76
5. Subsistema de Varias Peticiones a la Vez	77
5.1. Introducción	77
5.2. Objetivo del Caso de Estudio	78
5.3. Descripción del Problema Planteado	78
5.4. Descripción del Algoritmo Desarrollado	78
5.5. Applet Desarrollado	80
5.6. Datos y Ejecuciones	101
5.7. Resultados y Conclusiones	101
6. Hilos en Java	103
6.1. Concurrencia e Hilos con Java	103
6.2. Objetivo del Caso de Estudio	103
6.3. Descripción del Algoritmo Utilizado	104
6.4. Applet Desarrollado	104
6.5. Datos y Ejecuciones	121
6.6. Resultados y Conclusiones	123
7. Exclusión Mutua	124
7.1. Introducción	124
7.2. Objetivo	124
7.3. Introducción a la Sincronización en Sistemas Distribuidos	125
7.4. Exclusión Mutua	126
7.5. Un Algoritmo Centralizado	127
7.6. Applet Desarrollado	128
7.7. Datos y Ejecuciones	148
7.8. Resultados y Conclusiones	150
8. La Aplicación	151
9. Conclusiones y Líneas Futuras	154
Bibliografía	156
Índice alfabético	157

Índice de figuras

2.1. Pantalla Principal	22
2.2. Resultados Obtenidos	23
3.1. Pantalla Principal	44
3.2. Resultados Obtenidos	45
4.1. Pantalla Principal	75
4.2. Resultados Obtenidos	75
5.1. Pantalla Principal	101
5.2. Resultados Obtenidos	102
6.1. Pantalla de configuración	122
6.2. Pantalla de resultados	122
7.1. Pantalla Principal	149
7.2. Resultados Obtenidos	149
8.1. Página Principal	152
8.2. Menu Vertical y Carrusel de Imágenes	153

Capítulo 1

Introducción

Este Trabajo Final de Aplicación consiste en la investigación, análisis y desarrollo de los diferentes algoritmos de administración de recursos de Sistemas Operativos convencionales y distribuidos que se estudian en el laboratorio de la cátedra de Sistemas Operativos.

Este trabajo tiene como objetivo el desarrollo una aplicación web que contenga a los distintos algoritmos para ser ejecutados desde Internet y sean utilizado como herramienta de laboratorio por los alumnos de la cátedra.

Los distintos casos de estudio abarcados se han desarrollados en applets utilizando el lenguaje Java y como IDE a NetBeans, una poderosa herramienta para el desarrollo de aplicaciones orientadas a objetos y el desarrollo de applets.

Para la construcción del entorno web se trabajó con los lenguajes HTML y PHP, para el tratamiento de la información de los alumnos, se usó el motor de base de datos MySQL.

Las herramientas utilizadas para la construcción de la aplicación son libres y gratuitas por lo tanto no es necesario el uso de licencias para utilizar las mismas.

Los casos de estudio desarrollados son: Algoritmos de Planificación del Procesador, Anomalía Belady, Subsistema de Disco de Una Petición a la Vez, Subsistema de Disco de Varias Peticiones a la Vez, Hilos en Java y como así también el desarrollo de un algoritmo utilizado en la sincronización de procesos en sistemas operativos distribuidos, llamado Exclusión Mútua.

En los siguientes capítulos se desarrolla un breve marco teórico de cada caso de estudio, el código fuente de los applets y la descripción de cada una de las funcionalidades de los mismos.

Capítulo 2

Algoritmos de Planificación del Procesador

2.1. Introducción

Como parte de las estrategias de administración de recursos por parte del S. O. se puede considerar las de administración o asignación del procesador, es decir aquellas según las cuales los S. O. seleccionan a cuál de los procesos listos para ser ejecutados en ejecución concurrente, le asignarán el procesador en un momento dado, es decir, a qué proceso darán la posibilidad de utilizar la CPU para ejecutar sus propias instrucciones; a esta decisión también se la conoce como despacho del proceso.

2.2. Objetivo del Caso de Estudio

El objetivo del presente caso consistió en la realización de un applet que implemente las principales estrategias de asignación del procesador a procesos en ejecución concurrente.

El applet desarrollado muestra los datos resultantes de la simulación en una pantalla dentro del mismo, en la cual se detalla para cada simulación efectuada, un resumen de los datos obtenidos pertenecientes a cada una de las estrategias consideradas, lo que permite su análisis y discusión posterior.

2.3. Descripción del Problema Planteado

El concepto central de cualquier Sistema Operativo es el de *proceso*: una abstracción de un programa en ejecución también llamada *tarea* [1].

En sistemas de *multiprogramación* la CPU alterna de programa en programa, en un esquema de *seudoparalelismo*, es decir que la cpu ejecuta en cierto instante un solo programa, intercambiando muy rápidamente entre uno y otro [1].

El *paralelismo real de hardware* se da en las siguientes situaciones:

- En ejecución de instrucciones de programa con más de un procesador de instrucciones en uso simultáneamente.
- Con la superposición de ejecución de instrucciones de programa con la ejecución de una o más operaciones de entrada / salida.

El objetivo es aumentar el paralelismo en la ejecución.

El *modelo de procesos* posee las siguientes características:

- Todo el software ejecutable, inclusive el Sistema Operativo, se organiza en varios *procesos secuenciales* o *procesos*.
- Un proceso incluye al programa en ejecución y a los valores activos del contador, registros y variables del mismo.
- Conceptualmente cada proceso tiene su propia CPU virtual.
- Si la CPU alterna entre los procesos, la velocidad a la que ejecuta un proceso no será uniforme.
- Un proceso es una actividad de un cierto tipo, que tiene un programa, entrada, salida y estado.
- Un solo procesador puede ser compartido entre varios procesos con cierto *algoritmo de planificación*, el cual determina cuándo detener el trabajo en un proceso y dar servicio a otro distinto.

En cuanto a las *jerarquías de procesos* es necesario señalar que los Sistemas Operativos deben disponer de una forma de crear y destruir procesos cuando se requiera durante la operación, teniendo además presente que los procesos pueden generar procesos hijos mediante llamadas al Sistema Operativo, pudiendo darse la ejecución en paralelo.

Respecto de los *estados del proceso* deben efectuarse las siguientes consideraciones:

- Cada proceso es una entidad independiente pero frecuentemente debe interactuar con otros procesos.
- Los procesos pueden bloquearse en su ejecución porque:
 - Desde el punto de vista lógico no puede continuar porque espera datos que aún no están disponibles.
 - El Sistema Operativo asignó la CPU a otro proceso.

Los estados [2] [3] que puede tener un proceso son:

- En ejecución: utiliza la CPU en el instante dado.
- Listo: ejecutable, se detiene en forma temporal para que se ejecute otro proceso.
- Bloqueado: no se puede ejecutar debido a la ocurrencia de algún evento externo.

Son posibles cuatro *transiciones* entre estos estados.

2.4. Estados de Procesos

Durante su existencia un proceso pasa por una serie de estados discretos, siendo varias las circunstancias que pueden hacer que el mismo cambie de estado.

Debido a ello se puede establecer una *Lista de Listos* para los procesos *listos* y una *Lista de Bloqueados* para los *bloqueados* [4].

La *Lista de Listos* se mantiene en orden prioritario y la *Lista de Bloqueados* está desordenada, ya que los procesos se desbloquean en el orden en que tienen lugar los eventos que están esperando.

Al admitirse un trabajo en el sistema se crea un proceso equivalente y es insertado en la última parte de la *Lista de Listos*.

La asignación de la CPU al primer proceso de la *Lista de Listos* se denomina *Despacho*, que es ejecutado por una entidad del Sistema Operativo llamada *Despachador* [1] [2].

El *Bloqueo* es la única transición de estado iniciada por el propio proceso del usuario, puesto que las otras transiciones son iniciadas por entidades ajenas al proceso.

Los sistemas que administran los procesos deben poder crear, destruir, suspender, reanudar, cambiar la prioridad, bloquear, despertar y despachar un proceso.

La *creación* [1] de un proceso significa:

- Dar nombre al proceso.
- Insertar un proceso en la lista del sistema de procesos conocidos.
- Determinar la prioridad inicial del proceso.
- Crear el bloque de control del proceso.
- Asignar los recursos iniciales del proceso.

Un proceso puede *crear un nuevo proceso*, en cuyo caso el proceso creador se denomina *proceso padre* y el proceso creado *proceso hijo* y se obtiene una *estructura jerárquica de procesos* [1].

La *destrucción* [1] de un proceso implica:

- Borrarlo del sistema.
- Devolver sus recursos al sistema.
- Purgarlo de todas las listas o tablas del sistema.

- Borrar su bloque de control de procesos.

Un proceso *suspendido* no puede proseguir hasta que otro proceso lo reanude.

Reanudar (reactivar) un proceso implica reiniciarlo en el punto donde fue suspendido.

La *destrucción* de un proceso puede o no significar la destrucción de los procesos hijos, según el Sistema Operativo.

Generalmente se denomina *Tabla de Procesos* al conjunto de información de control sobre los distintos procesos.

Una *interrupción* es un evento que altera la secuencia en que el procesador ejecuta las instrucciones; es un hecho generado por el hardware del computador.

Cuando ocurre una interrupción, el Sistema Operativo:

- Obtiene el control.
- Salva el estado del proceso interrumpido, generalmente en su bloque de control de procesos.
- Analiza la interrupción.
- Transfiere el control a la rutina apropiada para la manipulación de la interrupción.

Una interrupción puede ser iniciada por un proceso en estado de ejecución o por un evento que puede o no estar relacionado con un proceso en ejecución.

Generalmente las *interrupciones* se pueden clasificar en los siguientes *tipos*:

- ● *SVC (llamada al supervisor)*: es una petición generada por el usuario para un servicio particular del sistema.
- ● *Entrada / Salida*: son iniciadas por el hardware de Entrada / Salida.
- ● *Externas*: son causadas por distintos eventos, por ejemplo, expiración de un cuanto en un reloj de interrupción o ecepción de una señal de otro procesador en un sistema multiprocesador.

- *De reinicio*: ocurren al presionar la *tecla de reinicio* o cuando llega una instrucción de reinicio de otro procesador en un sistema multiprocesador.
- *De verificación de programa*: son causadas por errores producidos durante la ejecución de procesos.

2.5. Descripción de los Algoritmos Utilizados

El applet desarrollado en Java, implementa cuatro estrategias de planificación del procesador, las cuales son:

- **FIFO**: Primero en llegar, primero en ser despachado, es decir que los procesos son atendidos según su orden en la lista de procesos listos y una vez que reciben el procesador lo utilizan hasta que finalizan o hasta que se presenta una petición de entrada / salida requerida por el propio programa [1].
- **RR**: Round Robin: Los procesos son atendidos según su orden en la lista de procesos listos, pero disponen de un tiempo limitado (quantum) del procesador, es decir que pueden ser interrumpidos por requerimientos propios de entrada / salida o por haber agotado su tiempo de procesador; obviamente que la otra causa de interrupción es la finalización del proceso [1].
- **HRN**: Los procesos son atendidos según su prioridad en la lista de procesos listos; la prioridad depende de la relación de respuesta: $(TE + TS) / TS$, donde $TE =$ Tiempo de Espera y $TS =$ Tiempo de Servicio; es decir que un proceso tiene mayor probabilidad de acceder a la CPU si ha hecho poco uso de ella; el tiempo de espera más el tiempo de servicio es el tiempo total que lleva el proceso en el sistema, es decir la cantidad de ciclos de control que se han contabilizado en la simulación, en tanto que el tiempo de servicio es el número de ciclos de control que el proceso ha utilizado; en este caso el proceso que dispone de la CPU puede ser interrumpido porque finaliza o porque requiere una operación de entrada / salida [1].
- **RNM**: Los procesos son atendidos según su nivel en la lista de procesos listos, la cual se divide en subcolas, cada una de ellas asociada a

un nivel determinado, pero los procesos disponen de un tiempo limitado (cuantum) del procesador; si son interrumpidos por entrada / salida permanecen en la subcola del nivel donde están, pero si son interrumpidos por tiempo pasan a un nivel mayor, que tiene menor preferencia, que es atendido cuando ya no hay procesos listos en los niveles inferiores; de allí el nombre de Retroalimentación de Niveles Múltiples; de lo antes mencionado se desprende que un proceso puede ser interrumpido porque finaliza, porque agota su tiempo de procesador o porque requiere una operación de entrada / salida. En la implementación efectuada los niveles identificados con un número menor son los que tienen de hecho mayor prioridad y los identificados con un número mayor son los que reciben menor prioridad [1].

2.6. Applet Desarrollado

El código fuente del programa desarrollado es el siguiente:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ProcesoEnApplet extends javax.swing.JApplet {
    public void init() {
        this.resize(675, 300);
        initComponents()
    }
    private void initComponents() {
        jScrollPane1 = new javax.swing.JScrollPane();
        salida = new javax.swing.JTextArea();
        tfNroProcesos = new javax.swing.JTextField();
        tfNroCiclos = new javax.swing.JTextField();
```

```
jLabel1 = new javax.swing.JLabel();
jLabel2 = new javax.swing.JLabel();
chkFIFO = new javax.swing.JCheckBox();
chkRR = new javax.swing.JCheckBox();
chkHRN = new javax.swing.JCheckBox();
chkRNM = new javax.swing.JCheckBox();
btnEjecutar = new javax.swing.JButton();
btnWhoIs = new javax.swing.JButton();
btnLimpiar = new javax.swing.JButton();
salida.setEditable(false);
salida.setFocusable(false);
salida.setColumns(40);
salida.setRows(15);
salida.setTabSize(4);
salida.setDoubleBuffered(true);
jScrollPane1.setViewportView(salida);
tfNroProcesos.setColumns(2);
tfNroProcesos.setText("5");
tfNroProcesos.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
tfNroProcesos.setToolTipTextText
("Ingrese la cantidad de procesos a utilizar en la simulaci?n.");

tfNroCiclos.setColumns(2);
tfNroCiclos.setText("20");
tfNroCiclos.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
```

```

tfNroCiclos.setToolTipText
("Se sugiere entre 30 y 40 ciclos de control por cada 10 procesos.");
jLabel1.setText("Nro. Procesos");
jLabel2.setText("Nro. Ciclos de Control");

chkFIFO.setText("FIFO");
chkFIFO.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
chkFIFO.setMargin(new java.awt.Insets(0, 0, 0, 0));
chkFIFO.setSelected(true);

chkRR.setText("RR");
chkRR.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
chkRR.setMargin(new java.awt.Insets(0, 0, 0, 0));
chkRR.setSelected(true);

chkHRN.setText("HRN");
chkHRN.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
chkHRN.setMargin(new java.awt.Insets(0, 0, 0, 0));
chkHRN.setSelected(true);

chkRNM.setText("RNM");
chkRNM.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
chkRNM.setMargin(new java.awt.Insets(0, 0, 0, 0));
chkRNM.setSelected(true);

btnEjecutar.setText("Ejecutar Simulacion");
btnEjecutar.addActionListener (new ActionListener(){
public void actionPerformed(ActionEvent e){
int nroProcesos = Integer.parseInt(tfNroProcesos.getText());

```

```

int nroCiclos = Integer.parseInt(tfNroCiclos.getText ());
boolean sFIFO = chkFIFO.isSelected();
boolean sRR=chkRR.isSelected ();
boolean sHRN=chkHRN.isSelected ();
boolean sRNM=chkRNM.isSelected ();
if(nroProcesos >0 && nroProcesos <=50){
if(nroCiclos > 0){
procesos = new Procesos();
i1 = salida.getText().length () + 738;
System.out.println(i1);
try{
procesos.simular
(nroCiclos, nroProcesos,salida, sFIFO, sRR, sHRN, sRNM);
}catch (Exception exception){
setContentPane (new JPanel());
getContentPane().add(new Label(".Error grave. "+
exception.getMessage () + " + exception.toString ());
getContentPane().repaint ();exception.printStackTrace ();
}finally{
salida.setCaretPosition(i1); } }
}else{
}}});
btnWhoIs.setText(."cerca de...");
btnWhoIs.addActionListener (new ActionListener(){

```

```

public void actionPerformed(ActionEvent e){
    JFrame ven = new JFrame();
    JTextArea jTextArea1= new JTextArea();
    jTextArea1.setBackground(new java.awt.Color(241, 243, 248));
    jTextArea1.setColumns(20);
    jTextArea1.setFont(new java.awt.Font("Bodoni MT", 0, 14));
    jTextArea1.setRows(5);
    jTextArea1.setText

```

(.Este Applet realiza la simulación de la Planificación del \nProcesador.

Fue desarrollado para la Catedra Sistemas \nOperativos, dictada por el Mgter.
David Luis La Red Martínez,

\nen la Facultad de Ciencias Exactas y Naturales y Agrimensura,
\n dependiente de la Universidad Nacional del Nordeste (UNNE)
,\n por los alumnos Nelson Fabián Rodríguez (Adscripto a la Catedra
)\ny Anibal Sebastian Rolando (Colaborador)");

```

jTextArea1.setEditable(false);
ven.setTitle("^cerca de...");
ven.add(jTextArea1);
ven.setVisible(true);
ven.setSize(400,200);
ven.setLocation(300,200);
ven.show();});});
btnLimpiar.setText("Limpiar");
btnLimpiar.addActionListener (new ActionListener(){
public void actionPerformed(ActionEvent e){

```

```

salida.setText ());});
javax.swing.GroupLayout layout = new javax.swing.GroupLayout
(this.getRootPane ().getContentPane ());
this.setLayout(layout);
layout.setHorizontalGroup(
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup())
.addContainerGap()
.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
475, javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup())
.addComponent(btnWhoIs, javax.swing
.GroupLayout.DEFAULT_SIZE, 162, Short.MAX_VALUE)
.addPreferredGap(javax.swing.LayoutStyle
.ComponentPlacement.RELATED))
.addGroup(layout.createSequentialGroup())
.addComponent(btnLimpiar, javax.swing
.GroupLayout.DEFAULT_SIZE, 162, Short.MAX_VALUE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED))
.addGroup(layout.createSequentialGroup())
.addComponent(btnEjecutar, javax.swing

```

```

.GroupLayout.DEFAULT_SIZE, 162, Short.MAX_VALUE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED))
.addGroup(layout.createSequentialGroup())
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup())
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup())
.addComponent(chkFIFO)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(chkRR)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(chkHRN))
.addComponent(jLabel1)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
.addComponent(chkRNM).addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.TRAILING, false)
.addComponent(tfNroCiclos, javax.swing.GroupLayout.Alignment.LEADING)
.addComponent(tfNroProcesos, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 17, Short.MAX_VALUE))))
.addComponent(jLabel2))

```

```

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
    javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
.addContainerGap(20, Short.MAX_VALUE));
layout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
    new java.awt.Component[] {btnEjecutar, btnLimpiar, btnWhoIs});
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
        layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(layout.createParallelGroup
            (javax.swing.GroupLayout.Alignment.TRAILING)
            .addComponent(jScrollPane1, javax.swing
                GroupLayout.Alignment.LEADING, javax.swing
                GroupLayout.DEFAULT_SIZE, 278, Short.MAX_VALUE)
            .addGroup(layout.createSequentialGroup())
            .addGroup(layout.createParallelGroup
                (javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel1)
                .addComponent(tfNroProcesos, javax.swing
                    GroupLayout.PREFERRED_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

```

```

.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel2)
.addComponent(tfNroCiclos, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addGap(18, 18, 18)
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(chkFIFO)
.addComponent(chkRR)
.addComponent(chkHRN)
.addComponent(chkRNM))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(btnEjecutar, javax.swing.GroupLayout.PREFERRED_SIZE, 41,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(btnLimpiar, javax.swing.GroupLayout.PREFERRED_SIZE, 52,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(btnWhoIs, javax.swing.GroupLayout.PREFERRED_SIZE, 48,
javax.swing.GroupLayout.PREFERRED_SIZE)))
.addContainerGap());
layout.linkSize(javax.swing.SwingConstants.VERTICAL,

```

```

        new java.awt.Component[]
        {btnEjecutar, btnLimpiar, btnWhoIs});
    }
    private javax.swing.JTextField tfNroCiclos;
    private javax.swing.JTextField tfNroProcesos;
    private javax.swing.JTextArea salida;
    private javax.swing.JButton btnEjecutar;
    private javax.swing.JButton btnLimpiar;
    private javax.swing.JButton btnWhoIs;
    private javax.swing.JCheckBox chkFIFO;
    private javax.swing.JCheckBox chkHRN;
    private javax.swing.JCheckBox chkRNM;
    private javax.swing.JCheckBox chkRR;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JScrollPane jScrollPane1;
    private Procesos procesos;
    private int i1;
    private AcercaDe ad;
    }

    public class Proceso {
        public static final int LISTO = 0; //Estado listo
        public static final int INTERRUMPIDO_ES = 1; //Estado interrumpido por
E/S

```

```
public static final int INTERRUMPIDO_TIEMPO = 3; //Estado interrumpido
por tiempo

public static final int TERMINADO = 2; //Estado terminado

public static final int FIFO = 0;

public static final int RR = 1;

public static final int HRN = 2;

public static final int RNM = 3;

public char id;

public int estado;

public int cuenta;

public int pri;

public int nivel;

Proceso(){

id = ' ';

estado = 0;

cuenta = 0;

pri = 0;

nivel = 0;

}

Proceso(char id){

this.id = id;

estado = 0;

cuenta = 0;

pri = 0;

nivel = 0;
```

```

    }

    public void siguienteEstado(String s){

        //Math.random() devuelve un número al .azar" mayor o igual a 0.0 y menor que
1.0

        int i1 = 0;
        int i2 = 0;
        if(s == "FIFO" || s == "HRN"){
            i1 = 2;
        }else if(s == RoundRobin" || s == RNM"){
            i1 = 3;
        }else{
            throw new IllegalArgumentException
("Los valores permitidos son: FIFO, HRN, RoundRobin y RNM.");
        }
        i2 = (int) (Math.random() * i1) + 1; //El cast (int) trunca.
        this.estado = i2;
    }

    public void continuaBloqueadoES() throws RuntimeException{
        int i = (int) (Math.random () * 4);
        if(i == Proceso.INTERRUMPIDO_TIEMPO){
            i = Proceso.LISTO;
        }
        if(i == Proceso.TERMINADO){
            i = Proceso.INTERRUMPIDO_ES;
        }
    }

```

```

if(i > 3){
throw new RuntimeException("Esta mal el random...");
}
this.estado = i;
}
public void continuaBloqueadoTiempo() throws RuntimeException{
int i = (int)(Math.random() * 4);
if(i == Proceso.INTERRUMPIDO_ES){
i = Proceso.LISTO;
}
if(i == Proceso.TERMINADO){
i = Proceso.INTERRUMPIDO_TIEMPO;
}
if(i > 3){
throw new RuntimeException("Esta mal el random...");
}
this.estado = i;
}
public void reinicio(){
this.estado = 0;
this.cuenta = 0;
this.pri = 0;
this.nivel = 0;
}

```

```
}

```

2.7. Datos y Ejecuciones

Los datos para realizar la simulación se seleccionan e introducen en la pantalla principal del applet, siendo opcional la selección del algoritmo de planificación.

Los datos antes mencionados están predeterminados pudiendo ser modificados por el usuario, éstos son el número de ciclos de control y el número de procesos que intervendrán en la simulación.

Los resultados detallados de las ejecuciones se muestran paso a paso en la pantalla principal del applet con un resumen al final de cada una de las estrategias de planificación. Ver figura 2.1 de la página 22 y figura 2.2 de la página 23.

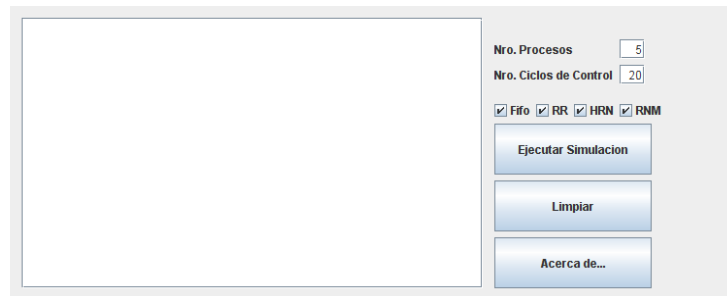


Figura 2.1: Pantalla Principal

2.8. Resultados y Conclusiones

Los resultados obtenidos con el applet desarrollado verifican lo indicado por la teoría con respecto a las estrategias de administración de recursos por parte de los Sistemas Operativos convencionales.

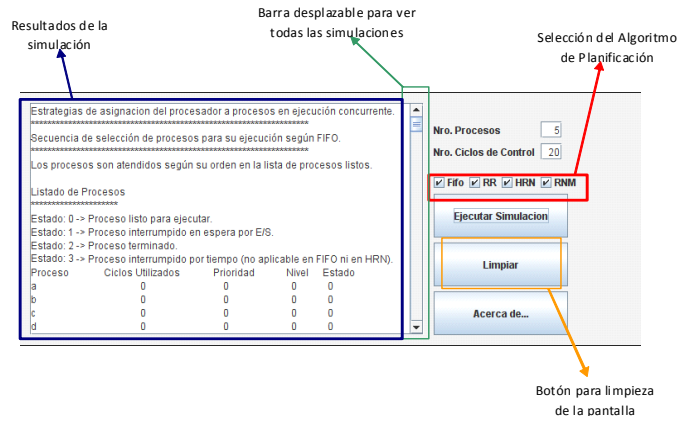


Figura 2.2: Resultados Obtenidos

El applet desarrollado posee una interfaz intuitiva lo cual facilita el uso del mismo y al permitir la elección de la estrategia a simular, facilita realizar un mejor análisis individual de cada una de las estrategias de administración planteadas.

Capítulo 3

Anomalía Belady

3.1. Introducción

Se ha desarrollado un applet aplicado a la simulación de requerimientos de paginación de memoria virtual partiendo desde los conceptos teóricos, utilizando arreglos y observando el comportamiento de los mismos evaluando si secuencias aleatorias de dichos requerimientos podrían llevar a situaciones del tipo de la llamada Anomalía de Belady o Anomalía FIFO.

3.2. Objetivo

Conforme a lo antes indicado, el objetivo del caso de estudio analizado fue el desarrollo de un applet en JAVA que efectuara el control de la posible Anomalía de Belady para una secuencia aleatoria de requerimientos de páginas en un sistema de paginación FIFO, visualizando en pantalla los datos obtenidos y gráficamente los principales resultados.

3.3. Reposición de Página por el Sistema de Primero en Entrar –Primero en Salir (FIFO)

3.3.1. Conceptos Generales de Gestión de Memoria

La organización y administración de la *memoria principal*, *memoria primaria* o *memoria real* de un sistema ha sido y es uno de los factores más importantes en el diseño de los S. O.

Los programas y datos deben estar en el almacenamiento principal para:

- Poderlos ejecutar.
- Referenciarlos directamente.

Históricamente el almacenamiento principal se ha considerado como un recurso costoso, por lo cual su utilización debía optimizarse.

Independientemente del esquema de organización hay que decidir las estrategias que se utilizarán para optimizar el rendimiento.

Las *estrategias de administración* deben considerar:

- ¿Cuándo se consigue un nuevo programa para colocar en la memoria?
- ¿Cuando el sistema lo pide específicamente o se intenta anticiparse a las peticiones?
- ¿Dónde se colocará el programa que se ejecutará a continuación?
- ¿Se prioriza el tiempo de carga o la optimización en el uso del almacenamiento?
- ¿Con qué criterio se desplazarán programas?

Las estrategias de administración del almacenamiento están dirigidas a la obtención del mejor uso posible del recurso del almacenamiento principal.

Se dividen en las siguientes **categorías**:

- Estrategias de búsqueda:
 - Estrategias de búsqueda por demanda.
 - Estrategias de búsqueda anticipada.
- Estrategias de colocación.
- Estrategias de reposición.

Las *estrategias de búsqueda* están relacionadas con el hecho de cuándo obtener el siguiente fragmento de programa o de datos para su inserción en la memoria principal.

En la *búsqueda por demanda* el siguiente fragmento de programa o de datos se carga al almacenamiento principal cuando algún programa en ejecución lo referencia.

Se considera que la *búsqueda anticipada* puede producir un mejor rendimiento del sistema.

Las *estrategias de colocación* están relacionadas con la determinación del lugar de la memoria donde se colocará (cargará) un programa nuevo.

Las *estrategias de reposición* están relacionadas con la determinación de qué fragmento de programa o de datos desplazar para dar lugar a los programas nuevos [1] [4].

- *Estrategias de búsqueda:*
 - Tratan de los casos en que una página o segmento deben ser traídos del almacenamiento secundario al primario.
 - Las estrategias de *búsqueda por demanda* esperan a que se haga referencia a una página o segmento por un proceso antes de traerlos al almacenamiento primario.
 - Los esquemas de *búsqueda anticipada* intentan determinar por adelantado a qué páginas o segmentos hará referencia un proceso para traerlos al almacenamiento primario antes de ser explícitamente referenciados [1].
- *Estrategias de colocación:*

- Tratan del lugar del almacenamiento primario donde se colocará una nueva página o segmento.
 - Los sistemas toman las decisiones de colocación de una forma trivial ya que una nueva página puede ser colocada dentro de cualquier marco de página disponible [1].
- *Estrategias de reposición:*
- Tratan de la decisión de cuál página o segmento desplazar para hacer sitio a una nueva página o segmento cuando el almacenamiento primario está completamente comprometido [1].

3.3.2. Reposición de Página por el Sistema FIFO

Se registra el momento en que cada página ingresa al almacenamiento primario.

Para reemplazar una página, se selecciona aquella que ha estado más tiempo almacenada.

Se presenta el inconveniente de que se pueden reemplazar páginas muy usadas, que serán llamadas de nuevo al almacenamiento primario casi de inmediato.

Se puede presentar la llamada *anomalía Belady* o *anomalía FIFO* [5]:

- Belady, Nelson y Shedler descubrieron que con la reposición FIFO, ciertos patrones de referencias de páginas causan más fallos de páginas cuando se aumenta el número de marcos (celdas) de páginas asignados a un proceso: en esto consiste la *anomalía FIFO*.
- Esta anomalía contradice a la intuición [1].

3.4. Applet Desarrollado

El applet desarrollado posee las siguientes características:

Recibe por pantalla los datos de configuración de la simulación referidos al número de celdas o marcos de página en la memoria real.

- También posee la opción de seleccionar la limpieza, o no, de la pantalla en cada ejecución de la simulación.

El vector de requerimiento se genera de forma automática y aleatoriamente, su tamaño es de tres veces la cantidad del número de celdas o marcos de páginas introducidos.

Las salidas por pantalla incluyen:

- Seguimiento del algoritmo de paginación FIFO, con indicación de los resultados que se van produciendo.
- Número de fallos de página ocurridos para cada test de la simulación, teniendo presente que para cada conjunto de datos de entrada, se efectúan tres simulaciones, una con el 80 % del número suministrado como número de celdas o marcos de página en la memoria real, otra con el 100 % de dicho número y una tercera con el 120 % del mismo.
- Se muestra información con respecto a si se ha producido o no la Anomalía de Belady en las simulaciones efectuadas.
- Representación gráfica de los principales resultados obtenidos, indicándose el número de fallos de páginas respecto del número de celdas de página, para las distintas simulaciones efectuadas.

El código del applet desarrollado es el siguiente:

```
package Rodriguez.Nelson.Fabian.Belady;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Graphics;

import javax.swing.JPanel;

public class Principal{

int fallos[] ;

int marcos [];
```

```
int requerimientos [];  
private int nm;  
String msg;  
Pantalla p;  
JPanel dib;  
Principal(Pantalla p, JPanel dib){  
fallos = new int[3];  
for(int i = 1; i < fallos.length; i++){  
fallos[i] = 0;  
}  
this.p = p;  
this.dib = dib;  
}  
public void setMarcos(int n){  
this.nm = n;  
p.println(.E1 nro de marcos es: "+ this.nm);  
}  
public void setRequerimientos(int n){  
this.requerimientos = new int[n];  
for(int i = 0; i < n; i++){  
this.requerimientos[i] = (int) (Math.random() * 50);  
}}  
public void simular(boolean borrarPantalla){  
String prefijo;
```

```
if(borrarPantalla){
    p.cls();
}
p.println("Vector de requerimientos de páginas para la simulacion:");
for(int i = 0; i < this.requerimientos.length; i++){
    p.print(String.valueOf(this.requerimientos[i] + ));
}
p.println();
for(int k = 0; k < 3; ++k){ // Ciclos de simulacion
    if(k==0){
        this.marcos = new int[(int)(nm * .8)];
        prefijo = "Primer Test: ";
    }else if (k == 1){
        this.marcos = new int[this.nm];
        prefijo = "Segundo Test: ";
    }else{
        this.marcos =new int[(int)(this.nm * 1.2)];
        prefijo = "Tercer Test: ";
    }
    for(int i = 0; i < this.requerimientos.length; i++){// Ciclo procesa requerimientos
        boolean encontro = false;
        for(int j = 0; j < this.marcos.length; j++){// Busco la página requerida en los
marcos
            if(this.marcos[j] == this.requerimientos[i]){
                encontro = true;
```

```
j = this.marcos.length;
}}
if(!encontre){ // Si la página no estaba cargada en los marcos
this.fallos[k]++; //Cuento el fallo
this.insertar(this.requerimientos[i]);
p.println("Fallo de página; se debe cargar a memoria real la página:
"+ this.requerimientos[i]);
}else{
p.println("Página encontrada en memoria real:" + this.requerimientos[i]);
}}
p.println(prefijo + this.fallos[k] + "fallos de página." );
p.println();
if(this.fallos [0]>= this.fallos [1]&& this.fallos [1]>= this.fallos [2]){
msg = ("No se produjo Anomalía en la simulación");
}else{
msg = ("Se produjo Anomalía Belady en la simulación");
}
p.datos(this.fallos[0],this.fallos[1],this.fallos[2],msg);
}
this.dibujar();
for(int ik = 0; ik < 3 ; ik++) {
this.fallos[ik]=0;
}}
private void dibujar() {
```

```
int x0,xN,y0,yN;

double xmin,xmax,ymin,ymax;

int apAncho,apAlto,apCero;

int[] posX = new int[3];

for(int i = 0; i < posX.length; i++){
posX[i] = (i + 1) * 50;
}

Graphics g = dib.getGraphics();

Dimension d = dib.getSize();

apAncho = d.width;

    apAlto = d.height;

g.setColor(Color.ORANGE);

g.fillRect(0, 0, apAncho, apAlto);

g.setColor(Color.black);

for (int i = 0; i < apAncho - 1; i = i + 5){

g.drawLine(i, 0, i, apAlto - 1);

}

for (int i = 0; i < apAlto - 1; i = i + 5){

g.drawLine(0, i, apAncho - 1, i);

}

double x = 0;

double y =0;

x0 = y0 = 0;

xN = apAncho-1;
```

```

yN = apAlto-1;

xmin = 0.0;
xmax = 50.0;
ymin = 0.0;
ymax = 50.0;

int yp;
int xp;
int xAnt, yAnt;
for(int i = 0; i < posX.length; i++){
g.setColor(Color.red);
xp = posX[i];
System.out.println("x: " + x + "xp: " + xp);
y = (double)this.fallos[i];
// Escalamos la coordenada y dentro de los limites de la ventana
yp = (int)(( y-ymin) * (apAlto-1) / (ymax-ymin) );
// Reconvertinos el valor cartesiano a punto de pantalla
yp = apAlto - yp;
g.fill3DRect(xp - 5 , yp, 10, apAlto - 1 , true);
g.setColor(Color.blue);
g.fillOval(xp, yp, 2, 2);
}}
private void insertar(int IDPagina) {
for(int i = this.marcos.length - 2; i >= 0 ; i-){ //Corre las páginas un marco a
la derecha

```

```
this.marcos[i + 1] = this.marcos[i]; // segun FIFO
}
this.marcos[0] = IDPagina; //Inserta la página en el primer marco
}}
package Rodriguez.Nelson.Fabian.Belady;
public class Pantalla extends javax.swing.JApplet {
private Principal p;
public void init() {
try {
java.awt.EventQueue.invokeLater(new Runnable() {
public void run() {
initComponents();
}});
} catch (Exception ex) {
ex.printStackTrace();
}
p = new Principal(this, this.jPanel1);
}
void cls() {
this.jTextArea1.setText();
}
void print(String s) {
this.jTextArea1.append(s);
}
```

```
void println(String s) {
    this.jTextArea1.append(s + "\n");
}

public void datos(int a, int b, int c, String d){
    this.jLabel8.setText(String.valueOf(a) );
    this.jLabel9.setText(String.valueOf(b) );
    this.jLabel10.setText(String.valueOf(c) );
    this.jLabel12.setText(d) ;
}

private void initComponents() {
    jScrollPane1 = new javax.swing.JScrollPane();
    jTextArea1 = new javax.swing.JTextArea();
    jLabel1 = new javax.swing.JLabel();
    jTextField1 = new javax.swing.JTextField();
    jButton1 = new javax.swing.JButton();
    jCheckBox1 = new javax.swing.JCheckBox();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();
    jLabel6 = new javax.swing.JLabel();
    jLabel7 = new javax.swing.JLabel();
    jLabel8 = new javax.swing.JLabel();
    jLabel9 = new javax.swing.JLabel();
}
```

```
jLabel10 = new javax.swing.JLabel();
jPanel1 = new javax.swing.JPanel();
jLabel11 = new javax.swing.JLabel();
jLabel12 = new javax.swing.JLabel();
jTextArea1.setColumns(20);
jTextArea1.setRows(5);
jScrollPane1.setViewportView(jTextArea1);
jLabel1.setText("Nro. de Marcos de Páginas");
jTextField1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1ActionPerformed(evt);
    }
});
jButton1.setLabel("Iniciar Simulación");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});
jCheckBox1.setSelected(true);
jCheckBox1.setText("Limpiar antes de iniciar la simulacion");
jLabel2.setText("80 % de marcos");
jLabel3.setText("100 % de marcos");
jLabel4.setText("120 % de marcos");
jLabel5.setText("FALLOS");
jLabel6.setText("FALLOS");
```

```
jLabel7.setText("FALLOS");
jLabel8.setText();
jLabel9.setText();
jLabel10.setText();
jPanel1.setBackground(new java.awt.Color(255, 255, 255));
jPanel1.setBorder(javax.swing
.BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0)));
javax.swing.GroupLayout jPanel1Layout = new javax.swing
.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
jPanel1Layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGap(0, 222, Short.MAX_VALUE));
jPanel1Layout.setVerticalGroup(
jPanel1Layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGap(0, 103, Short.MAX_VALUE));
javax.swing.GroupLayout layout = new javax.swing
.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
```

```
.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,  
layout.createSequentialGroup()).addGap(14, 14, 14)  
.addComponent(jScrollPane1, javax.swing  
.GroupLayout.PREFERRED_SIZE, 434,  
javax.swing.GroupLayout.PREFERRED_SIZE)  
.addGroup(layout.createParallelGroup(javax.swing  
.GroupLayout.Alignment.LEADING)  
.addGroup(layout.createSequentialGroup())  
.addGap(18, 18, 18)  
.addGroup(layout.createParallelGroup(javax.swing  
.GroupLayout.Alignment.LEADING)  
.addGroup(layout.createSequentialGroup())  
.addGroup(layout.createParallelGroup(javax.swing  
.GroupLayout.Alignment.LEADING)  
.addGroup(layout.createSequentialGroup())  
.addGap(10, 10, 10)  
.addGroup(layout.createParallelGroup(javax.swing  
.GroupLayout.Alignment.LEADING)  
.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,  
layout.createSequentialGroup()).addGroup  
(layout.createParallelGroup(javax.swing  
.GroupLayout.Alignment.LEADING)  
.addComponent(jLabel2)  
.addComponent(jLabel4)
```

```
.addComponent(jLabel3))
.addGap(10, 10, 10)
.addGroup(layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING, false)
.addComponent(jLabel10, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
.addComponent(jLabel9, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
.addComponent(jLabel8, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addGap(115, 115, 115))
.addComponent(jCheckBox1))
.addGroup(layout.createSequentialGroup())
.addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addGap(24, 24, 24))
.addGap(0, 0, 0)
.addComponent(jLabel11))
.addGroup(layout.createSequentialGroup())
.addComponent(jLabel1)
.addGap(18, 18, 18)
.addComponent(jTextField1, javax.swing
.GroupLayout.PREFERRED_SIZE, 41,
```

```
        javax.swing.GroupLayout.PREFERRED_SIZE))
    .addComponent(jButton1, javax.swing
    .GroupLayout.PREFERRED_SIZE, 155,
    javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel12, javax.swing
    .GroupLayout.PREFERRED_SIZE, 248,
    javax.swing.GroupLayout.PREFERRED_SIZE)))
    .addGroup(layout.createSequentialGroup())
    .addGap(228, 228, 228)
    .addGroup(layout.createParallelGroup(javax.swing
    .GroupLayout.Alignment.TRAILING, false)
    .addComponent(jLabel6).addComponent(jLabel7)
    .addComponent(jLabel5))).addGap(906, 906, 906));
    layout.setVerticalGroup(layout.createParallelGroup
    (javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup().addContainerGap()
    .addGroup(layout.createParallelGroup(javax.swing
    .GroupLayout.Alignment.TRAILING)
    .addComponent(jScrollPane1, javax.swing
    .GroupLayout.PREFERRED_SIZE, 318,
    javax.swing.GroupLayout.PREFERRED_SIZE)
    .addGroup(layout.createSequentialGroup().addGap(28, 28, 28)
    .addGroup(layout.createParallelGroup
    (javax.swing.GroupLayout.Alignment.TRAILING)
```

```
.addComponent(jTextField1, javax.swing
 GroupLayout.PREFERRED_SIZE, 20,
 javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jLabel1).addGap(6, 6, 6)
.addComponent(jButton1, javax.swing
 GroupLayout.PREFERRED_SIZE, 43,
 javax.swing.GroupLayout.PREFERRED_SIZE).addGap(7, 7, 7)
.addComponent(jCheckBox1).addGap(6, 6, 6)
.addGroup(layout.createParallelGroup(javax.swing
 GroupLayout.Alignment.LEADING)
.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
 layout.createSequentialGroup())
.addGroup(layout.createParallelGroup(javax.swing
 GroupLayout.Alignment.TRAILING)
.addGroup(layout.createSequentialGroup())
.addGroup(layout.createParallelGroup(javax.swing
 GroupLayout.Alignment.BASELINE)
.addComponent(jLabel2).addComponent(jLabel8))
.addPreferredGap(javax.swing.LayoutStyle
 ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup(javax.swing
 GroupLayout.Alignment.BASELINE)
.addComponent(jLabel3).addComponent(jLabel9))
.addPreferredGap(javax.swing.LayoutStyle
```

```
.ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel4).addComponent(jLabel10)))
.addGroup(layout.createSequentialGroup())
.addComponent(jLabel5)
.addPreferredGap(javax.swing.LayoutStyle
.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel6)
.addPreferredGap(javax.swing.LayoutStyle
.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel7))
.addPreferredGap(javax.swing.LayoutStyle
.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel12, javax.swing
.GroupLayout.PREFERRED_SIZE, 14
, javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle
.GroupLayout.Alignment.BASELINE)
.addComponent(jPanel1, javax.swing
.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing
.GroupLayout.PREFERRED_SIZE))
.addComponent(jLabel11, javax.swing
```

```
.GroupLayout.Alignment.TRAILING))))
.addContainerGap(49, Short.MAX_VALUE));}

private void jTextField1ActionPerformed
(java.awt.event.ActionEvent evt) {}

private void jButton1ActionPerformed
(java.awt.event.ActionEvent evt) {
p.setMarcos(Integer.parseInt( this(jTextField1.getText()));
p.setRequerimientos((int)(Integer.parseInt
(this(jTextField1.getText()) * 3));
p.simular(this(jCheckBox1.isSelected()));
}

private javax.swing.JButton jButton1;
private javax.swing.JCheckBox jCheckBox1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
```

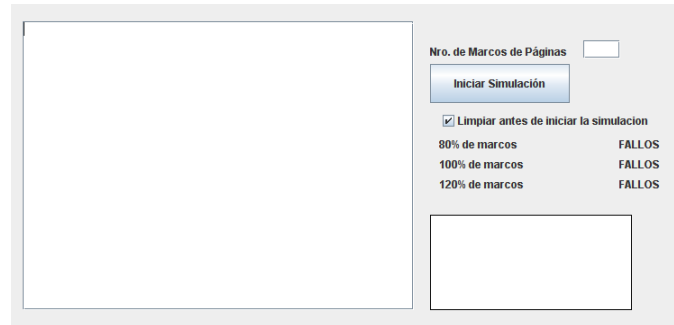


Figura 3.1: Pantalla Principal

```
private javax.swing.JLabel jLabel9;  
private javax.swing.JPanel jPanel1;  
private javax.swing.JScrollPane jScrollPane1;  
private javax.swing.JTextArea jTextArea1;  
private javax.swing.JTextField jTextField1;  
}
```

3.5. Datos y Ejecuciones

Los resultados detallados de las ejecuciones se muestran paso a paso en pantalla, también los resultados con la cantidad de fallos en cada simulación de acuerdo al porcentaje de celdas o marcos utilizados y la información si se produjo o no la anomalía Belady. Ver figura 3.1 de la página 44 y la figura 3.2 de la página 45.

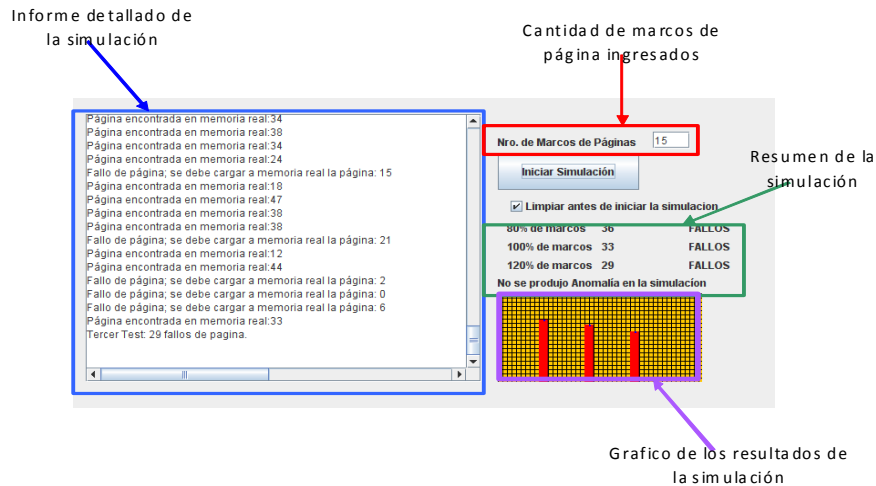


Figura 3.2: Resultados Obtenidos

3.6. Resultados y Conclusiones

Los resultados obtenidos con este applet ratifican lo previsto por la teoría de reposición de páginas por el sistema de Primero en Entrar –Primero en Salir (FIFO), en cuanto a que el fenómeno llamado Anomalia Belady o Anomalia FIFO es una situación especial que no se observa en ninguna de las simulaciones que se realizan, esto verifica lo que se puede pensar intuitivamente, que al aumentar la cantidad de celdas o marcos de página obtendremos menos fallos de página en un esquema de paginación FIFO.

También se ha observado que al aumentar considerablemente las celdas o marcos de página la cantidad de fallos no se reduce sino que tiende a ser constantes sin importar el porcentaje que utilicemos de ellas, esto es cercano a lo que se puede llamar una anomalía pero no llega a ser una anomalía (para producirse la anomalía no es suficiente con que el número de fallos de página se mantenga, sino que éste debe aumentar al aumentar el número de celdas de página disponibles en la memoria principal).

Cabe aclarar que estos resultados se obtienen con un número escaso de celdas o marcos de página, alejado a las situaciones reales en donde se utilizan centenas o miles de celdas de página disponibles.

Capítulo 4

Análisis del Rendimiento de un Subsistema de Disco de Una Petición a la Vez

4.1. Introducción

En este capítulo el caso de estudio desarrollado consiste en el Análisis del Rendimiento de un Subsistema de Disco de Una Petición a la Vez, el cual consta de un marco teórico del caso de estudio, el código fuente y la descripción de cada una de las ventanas del applet desarrollado.

4.2. Objetivo del Caso de Estudio

El objetivo consistió en la realización de un programa en JAVA que implementara el algoritmo de análisis de rendimiento para un Subsistema de Disco de Una Petición a la Vez.

También se estableció como objetivo la posibilidad de generar información detallada respecto de los cálculos efectuados con las distintas simulaciones y un análisis estadístico de los resultados logrados, como así también un gráfico

ilustrativo de los mismos.

4.3. Descripción del Problema Planteado

Si existe una población de clientes que demandan cierto servicio prestado por servidores:

Algunos clientes ingresarán a la red de colas y esperarán que un servidor quede disponible.

Algunas colas son:

Ilimitadas: pueden crecer tanto como sea necesario para contener a los clientes que esperan.

Limitadas: sólo pueden contener un número fijo de clientes en espera y quizás hasta ninguno.

Se deben tener en cuenta variables aleatorias que pueden ser descritas por *distribuciones probabilísticas*. [1] [2]

La variable aleatoria “q” representa el tiempo que emplea un cliente esperando en la cola a ser servido.

La variable aleatoria “s” representa la cantidad de tiempo que emplea un cliente en ser servido.

La variable aleatoria “w” representa el tiempo total que emplea un cliente en el sistema de colas: “ $w = q + s$ ”.

Fuente.

Los clientes son proporcionados a un sistema de colas desde una *fente* que puede ser infinita o finita.

Con una *fente infinita* la cola de servicio puede llegar a ser arbitrariamente grande.

Para una *fente finita* la cola de servicio es limitada.

Una fuente finita pero muy grande suele considerarse como infinita.

Llegadas.

Supondremos que los clientes llegan a un sistema de colas.

En los tiempos:

$$t_0 < t_1 < t_2 < \dots < t_n.$$

Los clientes llegan de uno en uno y nunca hay una colisión.

Las variables aleatorias “ τ_k ” miden los tiempos entre las llegadas sucesivas (arbitrario) y se denominan *tiempos entre Llegadas*:

Son variables aleatorias *independientes* y están *Idénticamente distribuidas*.

$$\tau_k = t_k - t_{k-1} \quad (k \geq 1).$$

Llegadas de Poisson.

Las llegadas pueden seguir distintos patrones arbitrarios.

Pero suele suponerse que forman un proceso de llegadas de Poisson:

- Los tiempos entre llegadas están distribuidos exponencialmente:

$$P(\tau \leq t) = 1 - e^{-\lambda t}$$

- La probabilidad de que lleguen exactamente n clientes en cualquier intervalo de longitud t es:

$$[e^{-\lambda t} (\lambda t)^n] / n! \quad (n = 0, 1, 2, \dots).$$

- λ es una tasa promedio de llegadas constante expresada en *clientes por unidad de tiempo*.
- El número de llegadas por unidad de tiempo se dice que tiene *distribución de Poisson* con una media λ . [1] [3]

Tiempos de servicio.

Se supone que los tiempos de servicio son aleatorios.

“ s_k ” es el tiempo de servicio que el k -ésimo cliente requiere del Sistema.

Un tiempo de servicio arbitrario se designa por “ s ”.

La distribución de los tiempos de servicio es:

- $W_s(t) = p(s \leq t)$.

Para un servicio aleatorio con una tasa promedio de servicio “ μ ”:

- $W_s(t) = p(s \leq t) = 1 - e^{-\mu t} \quad (t \geq 0)$.

Capacidad de la cola.

Las colas deben tener:

Capacidad infinita:

- Cada cliente que llegue puede entrar en el sistema de colas y esperar, independientemente de cuantos clientes hay en espera.

Capacidad cero (o sistemas de pérdidas):

- Los clientes que llegan cuando la instalación de servicio está ocupada no podrán ser admitidos el sistema.

Capacidad positiva:

- Los clientes que llegan solo esperan si hay lugar en la cola.

Numero de servidores en el sistema.

Los sistemas de colas se pueden categorizar según el número de Servidores en:

Sistemas de un solo servidor:

- Tienen un solo servidor y nada mas pueden darle servicio a un solo cliente a la vez.

Sistemas de servidores múltiples:

- Tienen “c” servidores con idéntica capacidad y pueden dar servicio a “c” clientes a la vez.

Resumen del Proceso de Poisson

Se define “ $p(k, t)$ ” como la probabilidad de exactamente “ k ” llegadas en un intervalo de tiempo de longitud “ t ”.

Un proceso es de Poisson si y solo si:

Para intervalos apropiadamente pequeños Δt :

- $P(k, t) = :$
 - $\lambda \Delta t$ para $k = 1$ (λ es la tasa promedio de llegadas).
 - $1 - \lambda \Delta t$ para $k = 0$.
 - 0 para $k > 1$.

Cualesquiera eventos definidos para tener lugar en intervalos de tiempo no superpuestos son mutuamente independientes.

Un proceso también es de Poisson si los tiempos entre llegadas sucesivas (*tiempos entre llegadas de primer orden*):

Son variables aleatorias exponenciales idénticamente distribuidas.

Si la variable aleatoria “ k ” indica el número de llegadas:

La *probabilidad* de, exactamente, “ k ” llegadas en un intervalo de longitud “ t ” es:

- $P(k, t) = \left[(\lambda t)^k e^{-\lambda t} \right] / k! \geq 0 \quad ; k = 0, 1, 2, \dots$

El *valor esperado* o *valor medio* de k es:

- $E(k) = \lambda t.$

La *varianza* de k es:

$$\blacksquare (\sigma_k) = \lambda t.$$

La suma de dos variables de Poisson aleatorias independientes “x” e “y” también describen un proceso de Poisson:

Los valores esperados son:

- $E(y) = \mu_2 = \lambda_2 t.$
- $E(x) = \mu_1 = \lambda_1 t.$

La probabilidad de “k” llegadas en el tiempo “t” es:

$$P(k, t) = [(\lambda_1 t + \lambda_2 t)^k e^{-(\lambda_1 t + \lambda_2 t)}] / k! \quad t \geq 0; \quad k = 0, 1, 2, \dots$$

$$P(k, t) = [(\mu_1 + \mu_2)^k e^{-(\mu_1 + \mu_2)}] / k!$$

$$P(k, t) = [\mu_s^k e^{-\mu_s}] / k! \quad \mu_s = \mu_1 + \mu_2$$

$$P(k, t) = [(\lambda_s t)^k e^{-\lambda_s t}] / k! \quad \lambda_s = \lambda_1 + \lambda_2$$

La suma de “n” procesos de Poisson independientes resulta en un proceso de Poisson con una tasa de llegada:

$$\lambda = \sum_{i=1}^n \lambda_i$$

Para un proceso de Poisson con una tasa de llegada “ λ ” se puede formar un nuevo proceso de Poisson utilizando *borradas aleatorias independientes*:

Cada llegada al proceso original:

- Se acepta al nuevo proceso con probabilidad “p”.
- Se rechaza con probabilidad “1 - P”.

La tasa de llegada del nuevo proceso derivado es “ λp ”.

La generalización para la descomposición de un proceso de Poisson en “ n ” procesos derivados independientes, cada uno con una probabilidad asociada “ p_i ” resulta:

$$\lambda_n = p_n \lambda.$$

$$\sum_{i=1}^n p_i = 1.$$

$$\sum_{i=1}^n \lambda_i = \sum_{i=1}^n p_i \lambda = \lambda \sum_{i=1}^n p_i = \lambda.$$

En un proceso de Poisson:

La probabilidad de que no haya llegadas en un intervalo de longitud “ t ” es:

$$\blacksquare P(0,t) = \left[(\lambda t)^0 e^{-\lambda t} \right] / 0! = e^{-\lambda t}.$$

La probabilidad de una o más llegadas en un intervalo de longitud “ t ” es:

$$1 - P(0,t) = 1 - e^{-\lambda t}.$$

La función de densidad de probabilidad para el *tiempo entre llegadas de primer orden (tiempo hasta la primera llegada)* es:

$$f_t(t) = \lambda e^{-\lambda t} \quad (t \geq 0).$$

El valor esperado “ t ” es:

$$\blacksquare E(t) = 1 / \lambda.$$

La varianza es:

$$\blacksquare (\sigma_t)^2 = 1 / \lambda^2$$

La función de densidad de probabilidad para el *tiempo entre llegadas de orden r -ésimo (tiempo hasta la r -ésima llegada)* es:

- $f_t(t) = (\lambda^r t^{r-1} e^{-\lambda t}) / (r-1)! \quad (t \geq 0, r = 1, 2, \dots)$

El valor esperado “ t ” es:

- $E(t) = r/\lambda.$

La desviación estándar es:

- $(\sigma_t)^2 = r/\lambda^2.$

Las instalaciones de servicio pueden proporcionar tiempos de servicio exponenciales:

La probabilidad de que el tiempo de servicio sea menor o igual a “ t ” es:

- $P(S \leq t) = 1 - e^{-\mu t} \quad (t \geq 0).$

La tasa promedio de servicio es “ μ ”.

El tiempo promedio de servicio es “ $1/\mu$ ”.

La función de densidad de probabilidad para el tiempo de servicio “ t ” es:

- $f_t(t) = \mu e^{-\mu t} \quad (t \geq 0).$

La media del tiempo de servicio es:

- $E(s) = 1/\mu.$

La varianza es “ $1/\mu^2$ ”.

Un servidor que opera de esta manera se denomina *servidor exponencial*. [1]

4.4. Descripción del Algoritmo Desarrollado

Se supone la siguiente situación:

Las peticiones de acceso a disco llegan como un proceso de Poisson con una tasa promedio de λ peticiones por minuto.

Si el disco está en uso, la petición se coloca en una cola primero en llegar, primero en ser servido.

Cuando el disco queda disponible se sirve la primera petición de la cola.

El tiempo de servicio es una variable aleatoria exponencialmente distribuida con un valor esperado de $1/\mu$ minutos.

La tasa promedio de servicio es de μ peticiones por minuto.

Se debe determinar, para cada uno de los casos:

- El valor esperado para el número total de peticiones al disco pendientes (en la cola o en servicio).
- Las probabilidades del estado límite.
- En este caso el dispositivo de disco contiene un solo brazo.
- Solo puede dar servicio a una petición a la vez.
- La tasa de servicio es μ .

4.5. Solución Propuesta

Si es el estado del sistema cuando hay i peticiones de disco al dispositivo de servicio de disco.

La tasa de llegadas de peticiones es independiente del estado del sistema:

- La probabilidad de la transición $s_i \rightarrow s_i + 1$ en el siguiente Intervalo de tiempo Δt es $\lambda\Delta t$.

Se considera al sistema como un proceso de nacimiento y muerte continuo de cadena sencilla y estados infinitos con:

- $d_i = 0 \quad i = 0.$
- $d_i = \mu \quad i = 1, 2, 3, \dots$
- $b_i = \lambda \quad i = 0, 1, 2, \dots$

Solo una petición puede ser servida en un momento dado y se sirve a una tasa μ .

$$\mu > \lambda$$

- Asegura que la longitud de la cola de peticiones en espera no crezca indefinidamente.

Se utilizan las relaciones:

$$P_{i+1} = (\lambda / \mu) P_i \quad i = 0, 1, 2, \dots$$

$$\sum_i P_i = 1.$$

$$P_1 = (\lambda / \mu) P_0.$$

$$P_2 = (\lambda / \mu) P_1 = (\lambda / \mu)^2 P_0.$$

$$P_i = (\lambda / \mu)^i P_0.$$

$$\sum_i P_i = 1 = \sum_i (\lambda / \mu)^i P_0 = 1 / [1 - (\lambda / \mu)] P_0.$$

$P_0 = 1 - (\lambda / \mu)$: probabilidad de que el sistema se encuentre ocioso.

$$P_i = (\lambda / \mu)^i P_0 = [1 - (\lambda / \mu)] (\lambda / \mu)^i. \quad i = 0, 1, 2, \dots$$

$P_i = [1 - (\lambda / \mu)] (\lambda / \mu)^i$: probabilidad que hayan i peticiones pendientes.

El número promedio de peticiones pendientes es:

$$E(i) = \sum_i i P^i = \sum_i i [1 - (\lambda / \mu)] (\lambda / \mu)^i = [1 - (\lambda / \mu)] \sum_i (\lambda / \mu)^i =$$

$$E(i) = [1 - (\lambda / \mu)] (\lambda / \mu) \sum_i (\lambda / \mu)^{i-1}$$

$$E(i) = [1 - (\lambda/\mu)] (\lambda/\mu) \{1/ [1 - (\lambda/\mu)^2]\}$$

$$E(i) = [(\lambda/\mu)\lambda/\mu] [1 - (\lambda/\mu)\lambda/\mu]^{-1}.$$

4.6. Applet Desarrollado

El código del programa desarrollado es el siguiente:

```

/*
(* TEORIA DE COLAS *)
(* Análisis del rendimiento de un subsistema de disco *)
(* Caso 1: El subsistema de disco solo puede dar servicio a una petición a la vez.
*)
(* Referencias y aclaraciones:
mu: Tasa promedio de servicio para un servidor.
mu = 1/Es(s)
Es(s): Tiempo de espera de servicio para un cliente.
lambda: Tasa promedio de llegadas de clientes al sistema de colas.
lambda = 1/Es(tau)
Es(tau): Tiempo de espera entre llegadas.
Es(tau) = 1/lambda
mu > lambda: Restricción para que la longitud de la cola de peticiones no crezca
indefinidamente.
tau: Intervalo entre llegadas.
Pn: Probabilidad de que haya "n" clientes en el sistema de colas en estado estable.
P0: Probabilidad de que el sistema se encuentre ocioso.
P0 = 1-(lambda/mu)

```

Pri: Probabilidad de que hayan "i" peticiones pendientes.

$$Pri = (1 - (\lambda/\mu))(\lambda/\mu)^i$$

(i=0,1,2,3,..)

Es(i) es el número promedio de peticiones pendientes.

$$Es(i) = (\lambda/\mu)(1 - (\lambda/\mu))^{(-1)}$$

*/

```
package subsistdiscounapet;

public class Analisis {

double tau,Pn,P0,muG,lambdaG,petPenG;

private java.util.ArrayList<Double> listaCoef;

private java.util.ArrayList<Double> listaPetPend;

private java.util.ArrayList<double[]> listaPuntos;

public Analisis(double mu,double lambda, int petPen){

this.listaCoef=new java.util.ArrayList();

this.listaPetPend= new java.util.ArrayList();

this.listaPuntos= new java.util.ArrayList<double[]>();

muG=mu; lambdaG=lambda; petPenG=petPen;

}

public String salida(){

this.getListCoef().clear();

this.getListPetPend().clear();

this.getListPuntos().clear();

String aux= ".Análisis del rendimiento de un subsistema \n"+

"de disco que solo puede dar servicio a una "+
```

```

"petición a la vez.\n"+
Colocar los valores máximos para mu, lambda
\ny el número de peticiones pendientes.
\n*****\n\n";
for(int i=10;i<=muG;i+=10)
for(int j=1; j<lambdaG;j+=2)
aux+= caso1(i,j,3);
aux+="\n\n*****";
aux+="\n***** RESUMEN FINAL *****";
aux+="\n*****";
aux+="\n\n Lista de Coeficientes Mu/Lambda:\n{";
for(int i=0;i<this.listaCoef.size();i++){
if(i%10==0 && i!=0) aux+="\n";
if(i!=(this.listaCoef.size()-1))
aux+=+((double)this.listaCoef.get(i))+";";
else
aux+=+((double)this.listaCoef.get(i))+"}";
}
aux+="\n\n Lista de Peticiones Pendientes:\n{";
for(int i=0;i<this.listaPetPend.size();i++){
if(i%10==0 && i!=0) aux+="\n";
if(i!=(this.listaPetPend.size()-1))
aux+=+((double)this.listaPetPend.get(i))+";";
else

```

```

aux+=+((double)this.listaPetPend.get(i))+"}";
}
aux+="\n\n Lista de Pares Ordenados:\n{";
for(int i=0;i<this.listaPuntos.size();i++){
if(i%10==0 && i!=0) aux+="\n";
if(i!=(this.listaPuntos.size()-1))
aux+="("+this.listaPuntos.get(i)[0]+", "+this.listaPuntos.get(i)[1]+")";
else
aux+="("+this.listaPuntos.get(i)[0]+", "+this.listaPuntos.get(i)[1]+")}";
}
return aux;
}
private double pri(double mu,double lambda,int i){
return ((1-(lambda*Math.pow(mu,-1))) * Math
.pow((lambda*Math.pow(mu,-1)),i));
}
private double es(double mu,double lambda){
return(lambda*Math.pow(mu,-1))* Math
.pow((1-(lambda*Math.pow(mu,-1))),(-1));
}
private String caso1(double mu,double lambda,int i){
String salida=;
double coef;
if((lambda/mu)<1){

```

```

salida+="Para que la cola no crezca indefinidamente
\n debe ser mayor MU que LAMBDA.\n";
coef=(lambda/mu);//TODO
salida+="Análisis de rendimiento de un subsistema de disco.\n";
salida+="Resultado del Caso1:\n";

salida+="E1 subsistema de disco solo puede dar\n servicio a una petición a la
vez.\n";

salida+="Los valores de mu, lambda y el coeficiente\n son los siguientes:\n";
salida+="mu+", "+lambda+", "+coef+"\n";
salida+="La cola no crecera eindefinidamente debido
\n a que Mu es mayor que Lambda.\n";
salida+="La probabilidad de tener 0,1,2,3,4,... peticiones\n";
salida+="pendientes son las siguientes: {";
for(int j=0;j<=i;j++)
if(j!=i)
salida+="pri(mu,lambda,j)+";
else
salida+="pri(mu,lambda,j);
salida += "}\n";

// salida+="pendientes son las siguientes:"+ pri(mu,lambda,i) +"\n";
salida+="E1 promedio de peticiones pendintes\n es el siguiente: "+es(mu,lambda)
+"\n";

this.getListaCoef().add(coef);
this.getListaPetPend().add(es(mu,lambda));
double[] punto=new double[2];

```

```
punto[0]=coef;
punto[1]=es(mu,lambda);
this.getListaPuntos().add(punto);
salida+="*****\n";
}
return salida;
}

public java.util.ArrayList getListaCoef() {
return listaCoef;
public java.util.ArrayList getListaPetPend() {
return listaPetPend;
}
public java.util.ArrayList<double[]> getListaPuntos() {
return listaPuntos;
}
}

package subsistdiscounapet;

public class AnalisisDeUnaPeticion extends javax.swing.JApplet {

public void init() {

try {

java.awt.EventQueue.invokeLaterAndWait(new Runnable() {

public void run() {

initComponents();

}});
```

```
    } catch (Exception ex) {
    ex.printStackTrace();
    }
}

private void initComponents() {
jDialog1 = new javax.swing.JDialog();
jLabel4 = new javax.swing.JLabel();
jButton2 = new javax.swing.JButton();
jScrollPane1 = new javax.swing.JScrollPane();
jTextArea1 = new javax.swing.JTextArea();
jLabel1 = new javax.swing.JLabel();
jTextField1 = new javax.swing.JTextField();
jLabel2 = new javax.swing.JLabel();
jTextField2 = new javax.swing.JTextField();
jTextField3 = new javax.swing.JTextField();
jLabel3 = new javax.swing.JLabel();
jPanel1 = new javax.swing.JPanel();
jButton1 = new javax.swing.JButton();
jLabel5 = new javax.swing.JLabel();
jLabel6 = new javax.swing.JLabel();
jLabel4.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel4.setText("jLabel4");
jButton2.setText("Aceptar");
jButton2.addActionListener(new java.awt.event.ActionListener() {
```

```
public void actionPerformed(java.awt.event.ActionEvent evt) {
    jButton2ActionPerformed(evt);
}});
javax.swing.GroupLayout jDialog1Layout = new javax.swing.GroupLayout
(jDialog1.getContentPane());
jDialog1.getContentPane().setLayout(jDialog1Layout);
jDialog1Layout.setHorizontalGroup(
    jDialog1Layout.createParallelGroup
        (javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jDialog1Layout.createSequentialGroup()
            .addGap(40, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGroup(jDialog1Layout.createParallelGroup
                (javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                    jDialog1Layout.createSequentialGroup()
                        .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 272,
                            javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(33, 33, 33))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                        jDialog1Layout.createSequentialGroup()
                            .addComponent(jButton2)
                            .addGap(134, 134, 134)))));
jDialog1Layout.setVerticalGroup(
    jDialog1Layout.createParallelGroup(javax.swing
```

```
.GroupLayout.Alignment.LEADING)
.addGroup(jDialog1Layout.createSequentialGroup()
.addContainerGap()
.addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 50,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle
.ComponentPlacement.UNRELATED)
.addComponent(jButton2)
.addContainerGap(32, Short.MAX_VALUE));
jTextArea1.setColumns(20);
jTextArea1.setEditable(false);
jTextArea1.setRows(5);
jTextArea1.setWrapStyleWord(true);
jScrollPane1.setViewportView(jTextArea1);
jLabel1.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel1.setText("Mu");
jTextField1.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField1.setText("40");
jLabel2.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel2.setText("Lambda");
jTextField2.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField2.setText("20");
jTextField3.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField3.setText("3");
```

```
jLabel3.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel3.setText("Peticones \nPendientes");
jPanel1.setBackground(new java.awt.Color(255, 255, 255));
jPanel1.setBorder(javax.swing.BorderFactory
.createLineBorder(new java.awt.Color(0, 0, 0)));
javax.swing.GroupLayout jPanel1Layout =
new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
jPanel1Layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
.addGap(0, 497, Short.MAX_VALUE));
jPanel1Layout.setVerticalGroup(
jPanel1Layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
.addGap(0, 217, Short.MAX_VALUE));
jButton1.setFont(new java.awt.Font("Tahoma", 1, 12));
jButton1.setText("Calcular");
jButton1.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt) {
jButton1ActionPerformed(evt);}});
jLabel5.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel5.setText("Detalles de los calculos realizados");
jLabel6.setFont(new java.awt.Font("Tahoma", 1, 14));
```



```
.addComponent(jLabel1)
.addComponent(jLabel3)
.addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 83,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addGap(21, 21, 21)
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING, false)
.addComponent(jTextField3)
.addComponent(jTextField1)
.addComponent(jTextField2, javax.swing.GroupLayout.DEFAULT_SIZE,
73, Short.MAX_VALUE)))
.addComponent(jLabel5)
.addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addContainerGap
(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE));
layout.setVerticalGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING, false)
.addGroup(layout.createSequentialGroup()
.addGap(17, 17, 17)
```

```
.addComponent(jLabel5)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(jScrollPane1, javax.swing
.GroupLayout.PREFERRED_SIZE, 207,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addGroup(layout.createSequentialGroup())
.addGap(57, 57, 57)
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.TRAILING)
.addGroup(layout.createSequentialGroup())
.addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addGap(67, 67, 67))
.addGroup(layout.createSequentialGroup())
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.BASELINE).addComponent(jLabel1)
.addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addGap(28, 28, 28)
.addComponent(jLabel2)
.addGap(37, 37, 37)
.addGroup(layout.createParallelGroup
```

```
(javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED_SIZE, 33,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
.addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 31,
javax.swing.GroupLayout.PREFERRED_SIZE)))
.addPreferredGap(javax.swing.LayoutStyle
.ComponentPlacement.UNRELATED).addComponent(jLabel6)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE).addGap(33, 33, 33));
}
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
double mu, lambda; int i;
try{
mu=Integer.parseInt(this(jTextField1.getText()));
lambda=Integer.parseInt(this(jTextField2.getText()));
if(mu>lambda){
i=Integer.parseInt(this(jTextField3.getText()));
```

```
Analisis a = new Analisis(mu,lambda,i);
this.jTextArea1.setText(a.salida());
this.listaValores=a.getListaPuntos();dibujar();
}else{
this.jLabel4.setText("Mu debe ser mayor a Lambda");
this.jDialog1.setSize(300,180);
this.jDialog1.setLocation(300,200);
this.jDialog1.setVisible(true);
}
} catch(Exception e){
this.jLabel4.setText(".Error en los parametros");
this.jDialog1.setSize(300,180);
this.jDialog1.setLocation(300,200);
this.jDialog1.setVisible(true);
}}
private void jButton2ActionPerformed
(java.awt.event.ActionEvent evt) {
this.jDialog1.hide();
}
private void dibujar() {
double XMax=-10000;
double YMax=-10000;
for(int i=0;i<this.listaValores.size();i++){
if(XMax < this.listaValores.get(i)[0])
```

```
XMax=this.listaValores.get(i)[0];
if(YMax < this.listaValores.get(i)[1])
YMax=this.listaValores.get(i)[1];
}
java.awt.Graphics g = jPanel1.getGraphics();
java.awt.Dimension d = jPanel1.getSize();
apAncho = d.width;
apAlto = d.height;
g.setColor(java.awt.Color.WHITE);
g.fillRect(0, 0, apAncho, apAlto);
g.setColor(java.awt.Color.BLACK);
g.drawRect(0, 0, apAncho-1, apAlto-1);
xmin = -0.005;
xmax = (XMax+0.03);
ymin = -1.5;
ymax = (YMax+0.03);
int apCero=(int)( (0-xmin) * (apAncho-1) / (xmax-xmin) );
int ypCero= (int)( (-0.3-ymin) * (apAlto-1) / (ymax-ymin) );
ypCero = apAlto - ypCero;
g.setColor(java.awt.Color.black);
g.drawLine(0,ypCero,(int)apAncho,ypCero);
g.drawLine(apCero,apAlto,apCero,0);
int xp,yp;
xp=(int)( ( 0.5 - xmin) * (apAncho-1) / (xmax-xmin) );
```

```
g.drawLine(apCero, ypCero, xp, ypCero);
for(int i=0;i<this.listaValores.size();i++){
double XP;
XP=this.listaValores.get(i)[0];
xp=(int)(( XP - xmin) * (apAncho-1) / (xmax-xmin) );
yp=(int)ValorY1(i);
g.setColor(java.awt.Color.orange);
g.fillOval(xp, yp, 7, 7);
}
double XP=0;
while(XP < XMax){
xp=(int)(( ( XP - xmin) * (apAncho-1) / (xmax-xmin) );
g.setColor(java.awt.Color.BLUE);
String s="+XP";
s=s.substring(0,3);
g.drawString(s,(int)xp,(int)ypCero+10);
XP +=0.2;
}
int YP=0;
while(YP <YMax){
g.setColor(java.awt.Color.BLUE);
String s="+YP";
g.drawString(s,(int)apCero,(int)ValorY2(YP));
YP +=1;
```

```
    }}  
    private int ValorY1(int valor) {  
        double x, y;  
        int retorno;  
        x = (valor * (xmax - xmin) / (apAncho - 1)) + xmin;  
        y=this.listaValores.get(valor)[1];  
        retorno = (int) ((y - ymin) * (apAlto - 1) / (ymax - ymin));  
        retorno = apAlto - retorno;  
        return (retorno);  
    }  
    private int ValorY2(double valor) {  
        double y;  
        int retorno;  
        y=valor;  
        retorno = (int) ((y - ymin) * (apAlto - 1) / (ymax - ymin));  
        retorno = apAlto - retorno;  
        return (retorno);  
    }  
    private javax.swing.JButton jButton1;  
    private javax.swing.JButton jButton2;  
    private javax.swing.JDialog jDialog1;  
    private javax.swing.JLabel jLabel1;  
    private javax.swing.JLabel jLabel2;  
    private javax.swing.JLabel jLabel3;
```

```
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTextArea jTextArea1;
private javax.swing.JTextField jTextField1;
private javax.swing.JTextField jTextField2;
private javax.swing.JTextField jTextField3;
private int apAlto,apAncho;
private double xmax,xmin,ymax,ymin;
private java.util.ArrayList<double[]> listaValores;
}
```

4.7. Datos y Ejecuciones

Al ejecutar el applet, se comienza con unos datos predefinidos de μ (μ), λ (λ) y las peticiones pendientes, estos datos pueden ser modificados por el usuario siempre que sean válidos de acuerdo a la teoría de colas.

También se podrá ver los resultados obtenidos en cada paso de la simulación en el panel superior de la ventana principal y un gráfico estadístico en el panel inferior de la misma ventana. Ver figura 4.1 en la página 75 y la figura 4.2 en la página 75.

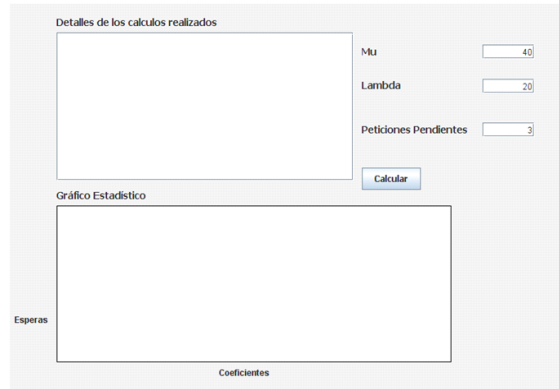


Figura 4.1: Pantalla Principal

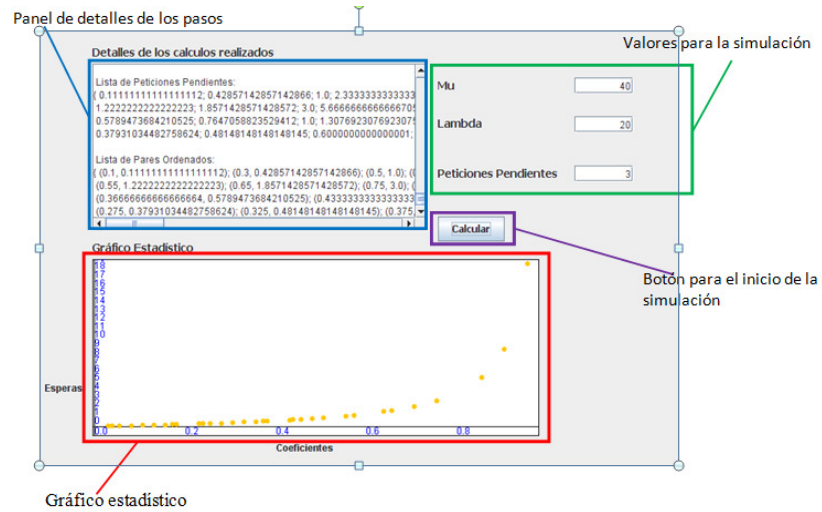


Figura 4.2: Resultados Obtenidos

4.8. Resultados y Conclusiones

Los resultados obtenidos con el applet desarrollado son coherentes con las previsiones teóricas, en cuanto a las diferencias en los tiempos en cola de espera en disco de las distintas peticiones, según las distintas cargas de trabajo simuladas.

La modalidad implementada de mostrar los resultados paso a paso por pantalla permite observar el comportamiento del algoritmo y facilita la comprensión de su funcionamiento.

Asimismo se observa un fuerte impacto en los tiempos de espera a partir de cierto nivel de carga de trabajo, expresada según los coeficientes considerados.

Capítulo 5

Análisis del Rendimiento de un Subsistema de Disco de Varias Peticiones a la Vez

5.1. Introducción

En este caso se desarrolló un software para la realización de un algoritmo que implemente el análisis de rendimiento de un subsistema de disco de varias peticiones a la vez, utilizando Teoría de Colas.

El estudio de las simulaciones efectuadas con el algoritmo señalado permite evaluar el comportamiento esperado de un subsistema de disco de varias peticiones a la vez cuando es sometido a distintas cargas de trabajo, es decir a conjuntos de peticiones de operaciones de acceso a los discos que configuran cantidades de trabajo distintas a ser atendidas.

La posibilidad de atender varias peticiones en paralelo requiere que el subsistema de disco disponga de varios brazos de acceso, que además sean independientes en sus movimientos, de tal manera que cada uno de ellos pueda atender a una petición independientemente.

5.2. Objetivo del Caso de Estudio

Como en los demás algoritmos, el objetivo del presente caso consiste en la realización de un programa en Java que implemente el algoritmo de análisis de rendimiento para el caso señalado.

Asimismo también se estableció como objetivo del caso, la inclusión en el programa de la posibilidad de generar información detallada respecto de los cálculos efectuados con las distintas simulaciones y un análisis estadístico de los resultados logrados, como así también un gráfico ilustrativo de los mismos.

5.3. Descripción del Problema Planteado

El problema que se plantea en este caso es similar al que se desarrolló en el caso de estudio del Análisis de Rendimiento de un Subsistema de Disco de Una Petición a la Vez, pero en este caso considerando varias peticiones a la vez.

5.4. Descripción del Algoritmo Desarrollado

Se supone la siguiente situación:

Las peticiones de acceso a disco llegan como un proceso de Poisson con una tasa promedio de λ peticiones por minuto.

Si el disco está en uso, la petición se coloca en una cola primero en llegar, primero en ser servido (FIFO).

Cuando el disco queda disponible se sirve la primera petición de la cola.

El tiempo de servicio es una variable aleatoria exponencialmente distribuida con un valor esperado de $1/\mu$ minutos.

La tasa promedio de servicio es de μ peticiones por minuto.

Se debe determinar, para cada uno de los casos:

- El valor esperado para el número total de peticiones al disco pendientes (en la cola o en servicio).

- Las probabilidades del estado límite.
- El dispositivo de disco contiene un número de brazos móviles.
- Cada brazo puede dar servicio a una petición de disco a la misma tasa μ .
- Se supone que un número infinito de peticiones pueden recibir servicio en paralelo.

Con i peticiones siendo servidas:

La probabilidad de que una petición en particular acabe siendo servida dentro del siguiente Δt es $\mu \Delta t$.

La probabilidad de que exactamente una petición cualquiera acabe es $i\mu\Delta t$ (buena aproximación de primer orden).

Cualquiera de las i peticiones puede terminar y provocar un cambio de estado. [1]

El sistema se ve como un proceso de nacimiento y muerte continuo de cadena sencilla y de estados infinitos con:

- $b_i = \lambda$ $i = 0, 1, 2, \dots$
- $d_i = 0$ $i = 0.$
- $d_i = i\mu$ $i = 1, 2, 3, \dots$

Ningún cliente tiene que esperar ya que se suponen infinitos servidores en paralelo.

Se utilizan las relaciones:

$$P_{i+1} = (b_i/d_{i+1}) P_i \quad i = 0, 1, 2, \dots$$

$$\sum_i P_i = 1.$$

$$P_1 = (\lambda/\mu)P_0.$$

$$P_2 = (\lambda/2\mu)P_1 = (1/2) (\lambda/\mu)^2 P_0.$$

$$P_3 = (\lambda/3\mu)P_2 = (1/(3,2)) (\lambda/\mu)^3 P_0.$$

$$P_i = (1/i!) (\lambda/\mu)^i P_0.$$

$$\sum_i P_i = 1 = \sum_i (1/i!) (\lambda/\mu)^i P_0.$$

$$\sum_i (x^n/n!) = e^x.$$

$$\sum_i P_i = 1 = \sum_i (1/i!) (\lambda/\mu)^i P_0 = e^{\lambda/\mu} P_0.$$

$$P_0 = e^{-\lambda/\mu}.$$

$$P_i = (\lambda/\mu)^i [(e^{-\lambda/\mu}) / i!].$$

$$E(i) = \sum_i iP_i = \sum_i i (\lambda/\mu)^i [(e^{-\lambda/\mu}) / i!] = (e^{-\lambda/\mu}) \sum_i i (\lambda/\mu) (1/i!) =$$

$$E(i) = (e^{-\lambda/\mu}) \sum_i (\lambda/\mu) (\lambda/\mu)^{i-1} [1/(i-1)!] =$$

$$E(i) = (e^{-\lambda/\mu}) (\lambda/\mu) \sum_i [1/(i-1)!] (\lambda/\mu)^{i-1} =$$

$$E(i) = (e^{-\lambda/\mu}) (\lambda/\mu) (e^{\lambda/\mu}) =$$

$$E(i) = (\lambda/\mu)$$

5.5. Applet Desarrollado

El código desarrollado es el siguiente:

```
package subsistdiscovariaspets;
```

```

public class Analisis {
double tau,Pn,P0,muG,lambdaG,petPenG;
private java.util.ArrayList<Double> listaCoef;
private java.util.ArrayList<Double> listaPetPend;
private java.util.ArrayList<double[]> listaPuntos;
public Analisis(double mu,double lambda, int petPen){
this.listaCoef=new java.util.ArrayList();
this.listaPetPend= new java.util.ArrayList();
this.listaPuntos= new java.util.ArrayList<double[]>();
muG=mu; lambdaG=lambda; petPenG=petPen;
}
public String salida(){
this.getListaCoef().clear();
this.getListaPetPend().clear();
this.getListaPuntos().clear();
String aux= ".Análisis del rendimiento de un subsistema \n"+
"de disco que puede dar servicio a varias "+
"peticiones a la vez.\n"+
"Colocar los valores máximos para mu, lambda
\ny el número de peticiones pendientes.
\n*****\n\n";
for(int i=10;i<=muG;i+=10)
for(int j=1; j<lambdaG;j+=2)
aux+= caso2(i,j,3);
}
}

```

```

aux+="\n\n*****";
aux+="\n***** RESUMEN FINAL *****";
aux+="\n\n*****";
aux+="\n\n Lista de Coeficientes Mu/Lambda:\n{ ";
for(int i=0;i<this.listaCoef.size();i++){
if(i%10==0 && i!=0) aux+="\n";
if(i!=(this.listaCoef.size()-1))
aux+=+((double)this.listaCoef.get(i))+";";
else
aux+=+((double)this.listaCoef.get(i))+"}";
}
aux+="\n\n Lista de Peticiones Pendientes:\n{ ";
for(int i=0;i<this.listaPetPend.size();i++){
if(i%10==0 && i!=0) aux+="\n";
if(i!=(this.listaPetPend.size()-1))
aux+=+((double)this.listaPetPend.get(i))+";";
else
aux+=+((double)this.listaPetPend.get(i))+"}";
}
aux+="\n\n Lista de Pares Ordenados:\n{ ";
for(int i=0;i<this.listaPuntos.size();i++){
if(i%10==0 && i!=0) aux+="\n";
if(i!=(this.listaPuntos.size()-1))
aux+=(" "+this.listaPuntos.get(i)[0]+", "+this.listaPuntos.get(i)[1]+");";

```

```

else
aux+="("+this.listaPuntos.get(i)[0]+", "+this.listaPuntos.get(i)[1]+")";
}return aux;
}
private double pri(double mu,double lambda,int i){
return (Math.pow(lambda*Math.pow(mu, -1),i)*((Math.pow
(Math.E,(lambda*Math.pow(mu, -1))))*(Math.pow(factorial(i),-1))));
}
private int factorial(int i){
int aux=1;
for (int j=1;j<=i;j++)
aux=aux*j;
return aux;
}
private double es(double mu,double lambda){
return(lambda*Math.pow(mu,-1));
}
private String caso2(double mu,double lambda,int i){
String salida="";
double coef;
if((lambda/mu)<1){
salida+="Para que la cola no crezca indefinidamente
\n debe ser mayor MU que LAMBDA.\n";
coef=(lambda/mu);//TODO

```

```

salida+=.A analisis de rendimiento de un subsistema de disco.\n";
salida+=Resultado del Caso2:\n";

salida+=.E1 subsistema de disco puede dar\n servicio a varias peticiones a la
vez.\n";

salida+="Los valores de mu, lambda y el coeficiente\n son los siguientes:\n";
salida+=+mu+", "+lambda+", "+coef+"\n";

salida+="La cola no crecera eindefidamente debido\n a que Mu es mayor que
Lambda.\n";

salida+="La probabilidad de tener 0,1,2,3,4,... peticiones\n";

salida+="pendientes son las siguientes: {";

for(int j=0;j<=i;j++)
if(j!=i)
salida+=+pri(mu,lambda,j)+" ";
else
salida+=+pri(mu,lambda,j);

salida +="}\n";

// salida+="pendientes son las siguientes:"+ pri(mu,lambda,i) +"\n";

salida+=.E1 promedio de peticiones pendintes\n es el siguiente:

"+es(mu,lambda) +"\n";

this.getListCoef().add(coef);

this.getListPetPend().add(es(mu,lambda));

double[] punto=new double[2];

punto[0]=coef;

punto[1]=es(mu,lambda);

this.getListPuntos().add(punto);

```

```

salida+="*****\n";
}
return salida;
}
/*
 * @return the listaCoef
 */
public java.util.ArrayList getListaCoef() {
return listaCoef;
}
/*
 * @return the listaPetPend
 */
public java.util.ArrayList getListaPetPend() {
return listaPetPend;
}
/*
 * @return the listaPuntos
 */
public java.util.ArrayList<double[]> getListaPuntos() {
return listaPuntos;
}
}
package subsistdiscovariaspet;

```

```
import subsistdiscovariaspet.*;

public class AnalisisDeVariasPet extends javax.swing.JApplet {

    public void init() {

        try {

            java.awt.EventQueue.invokeLater(new Runnable() {

                public void run() {

                    initComponents();

                }

            });

        } catch (Exception ex) {

            ex.printStackTrace();

        }

        private void initComponents() {

            jDialog1 = new javax.swing.JDialog();

            jLabel4 = new javax.swing.JLabel();

            jButton2 = new javax.swing.JButton();

            jScrollPane1 = new javax.swing.JScrollPane();

            jTextArea1 = new javax.swing.JTextArea();

            jLabel1 = new javax.swing.JLabel();

            jTextField1 = new javax.swing.JTextField();

            jLabel2 = new javax.swing.JLabel();

            jTextField2 = new javax.swing.JTextField();

            jTextField3 = new javax.swing.JTextField();

            jLabel3 = new javax.swing.JLabel();

            jPanel1 = new javax.swing.JPanel();
```

```

jButton1 = new javax.swing.JButton();
jLabel5 = new javax.swing.JLabel();
jLabel6 = new javax.swing.JLabel();
jLabel7 = new javax.swing.JLabel();
jLabel8 = new javax.swing.JLabel();
jLabel4.setHorizontalAlignment
(javax.swing.SwingConstants.CENTER);
jLabel4.setText("jLabel4");
jButton2.setText("Aceptar");
jButton2.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt) {
jButton2ActionPerformed(evt);
}});
javax.swing.GroupLayout jDialog1Layout = new javax.swing
.GroupLayout(jDialog1.getContentPane());
jDialog1.getContentPane().setLayout(jDialog1Layout);
jDialog1Layout.setHorizontalGroup(
jDialog1Layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGroup(jDialog1Layout.createSequentialGroup()
.addGroup(jDialog1Layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(jDialog1Layout.createSequentialGroup()
.addContainerGap()

```

```

.addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 272,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addGroup(jDialog1Layout.createSequentialGroup())
.addGap(115, 115, 115)
.addComponent(jButton2))
.addContainerGap(19, Short.MAX_VALUE))
);
jDialog1Layout.setVerticalGroup(
jDialog1Layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGroup(jDialog1Layout.createSequentialGroup())
.addGap(26, 26, 26)
.addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE,
50, javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(jButton2).addContainerGap
(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE));
jTextArea1.setColumns(20);
jTextArea1.setEditable(false);
jTextArea1.setRows(5);
jTextArea1.setWrapStyleWord(true);
jScrollPane1.setViewportView(jTextArea1);
jLabel1.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel1.setText("Mu");

```

```

jTextField1.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField1.setText("40");
jLabel2.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel2.setText("Lambda");
jTextField2.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField2.setText("20");
jTextField3.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField3.setText("3");
jLabel3.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel3.setText("Peticiones \nPendientes");
jPanel1.setBackground(new java.awt.Color(255, 255, 255));
jPanel1.setBorder(javax.swing.BorderFactory
.createLineBorder(new java.awt.Color(0, 0, 0)));
javax.swing.GroupLayout jPanel1Layout =
new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
jPanel1Layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGap(0, 462, Short.MAX_VALUE));
jPanel1Layout.setVerticalGroup(
jPanel1Layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGap(0, 217, Short.MAX_VALUE));

```

```

jButton1.setFont(new java.awt.Font("Tahoma", 1, 12));
jButton1.setText("Calcular");
jButton1.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt) {
jButton1ActionPerformed(evt);
}});
jLabel5.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel5.setText("Detalles de los calculos realizados");
jLabel6.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel6.setText("Gráfico Estadístico");
jLabel7.setText("Esperas");
jLabel8.setText("Coeficientes");

javax.swing.GroupLayout layout = new javax.swing
.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addGroup(layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addGap(7, 7, 7)
.addComponent(jLabel7)

```

```

.addPreferredGap(javax.swing.LayoutStyle
.ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addComponent(jLabel5)
.addGroup(layout.createSequentialGroup())
.addComponent(jScrollPane1, javax.swing
.GroupLayout.PREFERRED_SIZE, 383,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle
.ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.TRAILING)
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
.addComponent(jLabel2)
.addComponent(jLabel1)
.addComponent(jLabel3))
.addComponent(jButton1, javax.swing
.GroupLayout.PREFERRED_SIZE, 83,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addGap(21, 21, 21).addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING, false)
.addComponent(jTextField3)

```

```

.addComponent(jTextField1)
.addComponent(jTextField2,
    javax.swing.GroupLayout.DEFAULT_SIZE, 73,
    Short.MAX_VALUE)))
.addComponent(jLabel6)
.addComponent(jPanel1, javax.swing
    GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE
    , javax.swing.GroupLayout.PREFERRED_SIZE)))
.addGroup(layout.createSequentialGroup())
.addGap(244, 244, 244)
.addComponent(jLabel8)))
.addContainerGap(javax.swing
    GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE));
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing
    GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup())
    .addGroup(layout.createParallelGroup(javax.swing
    GroupLayout.Alignment.LEADING, false)
    .addGroup(layout.createSequentialGroup())
    .addGap(17, 17, 17)
    .addComponent(jLabel5)
    .addPreferredGap(javax.swing.LayoutStyle

```

```

.ComponentPlacement.RELATED)
.addComponent(jScrollPane1, javax.swing
.GroupLayout.PREFERRED_SIZE, 207,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addGroup(layout.createSequentialGroup())
.addGap(57, 57, 57)
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.TRAILING)
.addGroup(layout.createSequentialGroup())
.addComponent(jTextField2, javax.swing
.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing
.GroupLayout.PREFERRED_SIZE)
.addGap(67, 67, 67))
.addGroup(layout.createSequentialGroup())
.addGroup(layout.createParallelGroup
(javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel1)
.addComponent(jTextField1, javax.swing
.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing
.GroupLayout.PREFERRED_SIZE))
.addGap(28, 28, 28)
.addComponent(jLabel2)

```

```

.addGap(37, 37, 37)

.addGroup(layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel3, javax.swing
.GroupLayout.PREFERRED_SIZE, 33,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jTextField3, javax.swing
.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))))
.addPreferredGap(javax.swing.LayoutStyle
.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
.addComponent(jButton1, javax.swing.GroupLayout
.PREFERRED_SIZE, 31,
javax.swing.GroupLayout.PREFERRED_SIZE)))
.addGap(11, 11, 11)

.addComponent(jLabel6)

.addPreferredGap(javax.swing.LayoutStyle
.ComponentPlacement.RELATED)

.addComponent(jPanel1, javax.swing
.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,

```

```

javax.swing.GroupLayout.PREFERRED_SIZE)
.addGap(8, 8, 8)
.addComponent(jLabel8)
.addContainerGap()
.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
.addContainerGap(368, Short.MAX_VALUE)
.addComponent(jLabel7)
.addGap(151, 151, 151));
}
private void jButton1ActionPerformed
(java.awt.event.ActionEvent evt) {
double mu, lambda;
int i;
try{
mu=Integer.parseInt(this.jTextField1.getText());
lambda=Integer.parseInt(this.jTextField2.getText());
if(mu>lambda){
i=Integer.parseInt(this.jTextField3.getText());
Análisis a = new Análisis(mu,lambda,i);
this.jTextArea1.setText(a.salida());
this.listaValores=a.getListaPuntos();
dibujar();
}else{

```

```

this.jLabel4.setText("Mu debe ser mayor a Lambda");
this.jDialog1.setSize(300,180);
this.jDialog1.setLocation(300,200);
this.jDialog1.setVisible(true);
}
} catch(Exception e){
this.jLabel4.setText("Error en los parametros");
this.jDialog1.setSize(300,180);
this.jDialog1.setLocation(300,200);
this.jDialog1.setVisible(true);
}}
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
this.jDialog1.hide();
}
private void dibujar() {
double XMax=-10000;
double YMax=-10000;
for(int i=0;i<this.listaValores.size();i++){
if(XMax < this.listaValores.get(i)[0])
XMax=this.listaValores.get(i)[0];
if(YMax < this.listaValores.get(i)[1])
YMax=this.listaValores.get(i)[1];
}
java.awt.Graphics g = jPanel1.getGraphics();

```

```

java.awt.Dimension d = jPanel1.getSize();

apAncho = d.width;

apAlto = d.height;

g.setColor(java.awt.Color.WHITE);

g.fillRect(0, 0, apAncho, apAlto);

g.setColor(java.awt.Color.BLACK);

g.drawRect(0, 0, apAncho-1, apAlto-1);

xmin = -0.2;

xmax = (XMax+0.03);

ymin = -0.2;

ymax = (YMax+0.03);

int apCero=(int)( (0-xmin) * (apAncho-1) / (xmax-xmin) );

int ypCero= (int)( (0-ymin) * (apAlto-1) / (ymax-ymin) );

ypCero = apAlto - ypCero;

g.setColor(java.awt.Color.black);

g.drawLine(0,ypCero,(int)apAncho,ypCero);

g.drawLine(apCero,apAlto,apCero,0);

int xp,yp;

//yp = (int)( (0.5-ymin) * (apAlto-1) / (ymax-ymin) );

xp=(int)( ( 0.5 - xmin) * (apAncho-1) / (xmax-xmin) );

g.drawLine(apCero, ypCero, xp, ypCero);

for(int i=0;i<this.listaValores.size();i++){

double XP;

XP=this.listaValores.get(i)[0];

```

```

xp=(int)( (XP - xmin) * (apAncho-1) / (xmax-xmin) );
yp=(int)ValorY1(i);
g.setColor(java.awt.Color.orange);
g.fillOval(xp, yp, 7, 7);
}
double XP=0;
while(XP < XMax){
xp=(int)( ( XP - xmin) * (apAncho-1) / (xmax-xmin) );
g.setColor(java.awt.Color.BLUE);
String s="+XP";
s=s.substring(0,3);
g.drawString(s,(int)xp,(int)ypCero+10);
XP +=0.2;
}
double YP=0;
while(YP < YMax){
g.setColor(java.awt.Color.BLUE);
String s="+YP";
s=s.substring(0,3);
g.drawString(s,(int)apCero,(int)ValorY2(YP));
YP +=0.2;
}}
private int ValorY1(int valor) {
double x, y;

```

```

int retorno;

x = (valor * (xmax - xmin) / (apAncho - 1)) + xmin;
y=this.listaValores.get(valor)[1];
retorno = (int) ((y - ymin) * (apAlto - 1) / (ymax - ymin));
retorno = apAlto - retorno;

return (retorno);
}

private int ValorY2(double valor) {
double y;

int retorno;

y=valor;

retorno = (int) ((y - ymin) * (apAlto - 1) / (ymax - ymin));

retorno = apAlto - retorno;

return (retorno);
}

// Variables declaration - do not modify
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JDialog jDialog1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;

```

```

private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTextArea jTextArea1;
private javax.swing.JTextField jTextField1;
private javax.swing.JTextField jTextField2;
private javax.swing.JTextField jTextField3;
// End of variables declaration
private int apAlto,apAncho;
private double xmax,xmin,ymax,ymin;
private java.util.ArrayList<double[]> listaValores;
}
package subsistdiscovariaspet;
public class Prueba {
public static void main(String arg[]){
Analisis a=new Analisis(40,20,3);
System.out.print(a.salida());
}}
package subsistdiscovariaspet;
public class Main {
public static void main(String[] args) {
AnalisisDeRend ven=new AnalisisDeRend();

```

```
ven.setVisible(true);
}}
```

5.6. Datos y Ejecuciones

Al ejecutar el applet, comenzamos con unos datos predefinidos de μ (μ) lambda (λ) y las peticiones pendientes, estos datos pueden ser modificados por el usuario siempre que sean validos de acuerdo a la teoría de colas.

También podremos ver los resultados obtenidos en cada paso de la simulación en el panel superior de la ventana principal y un gráfico estadístico en el panel inferior de la misma ventana. Ver la figura 5.1 de la página 101 y la figura 5.2 de la página 102.

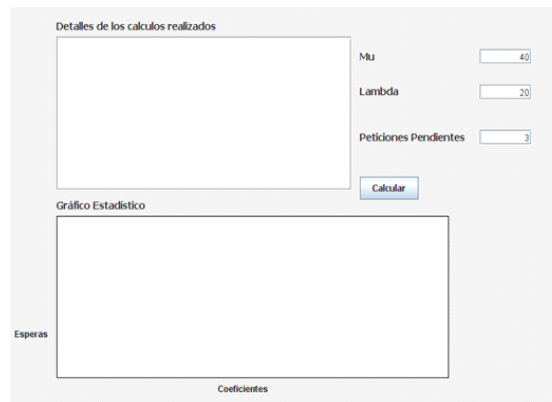


Figura 5.1: Pantalla Principal

5.7. Resultados y Conclusiones

Los resultados obtenidos con el applet desarrollado son similares a los del programa desarrollado para el SistOper como era de esperarse, además, se verificó lo visto en teoría en cuanto a las diferencias en tiempos en cola de espera en disco de las distintas peticiones, según las distintas cargas de trabajo simuladas.

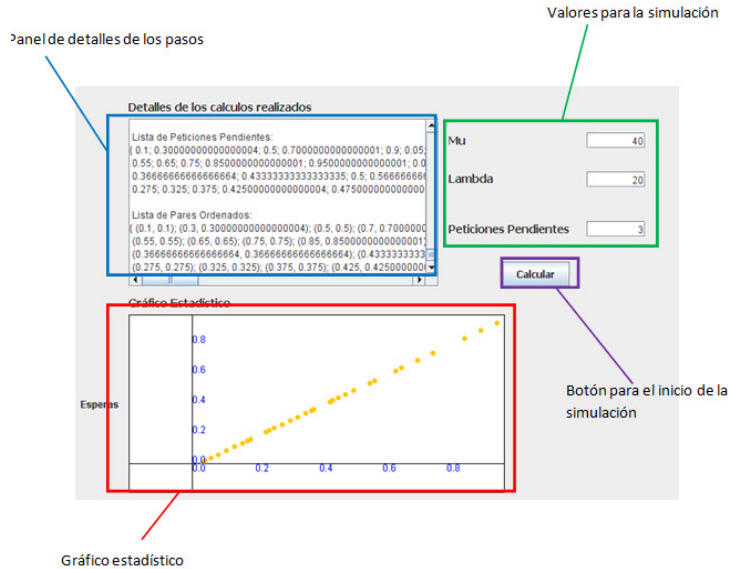


Figura 5.2: Resultados Obtenidos

La modalidad implementada de mostrar los resultados paso a paso por pantalla permite observar el comportamiento del algoritmo y facilita la comprensión de su funcionamiento.

Como era de esperarse, se observa que en todos los casos, la forma de la curva de los gráficos obtenidos es similar, apreciándose un muy buen ajuste de los valores obtenidos los cuales representados gráficamente se aproximan a una recta.

Asimismo se observa un leve impacto en los tiempos de espera ante incrementos de la carga de trabajo, expresada según los coeficientes considerados, observándose que el número promedio de peticiones pendientes tiende a 1.

Capítulo 6

Hilos en Java

6.1. Concurrencia e Hilos con Java

Los hilos o procesos ligeros son una parte de código o mini programa que puede ser ejecutado independientemente, de forma que una aplicación o un applet puede tener varios hilos ejecutándose simultáneamente y efectuando distintas tareas; estos hilos se encuentran dentro de un programa y son parte de él.

Los hilos, a veces también llamados contextos de ejecución, pueden ser utilizados para la implementación de algoritmos paralelos o procesos concurrentes, sin ser necesario disponer de equipos con estructura de multiprocesador. En el caso de un solo procesador, los procesos ligeros incorporan mecanismos para compartirlo, estableciéndose prioridades entre ellos y también facilidades de sincronización, cuando sea necesario [1].

6.2. Objetivo del Caso de Estudio

El objetivo fue desarrollar un applet que implementara el problema de *procesos productores y consumidores*, que permitiera generar en un arreglo el seguimiento de dicha simulación para posteriormente efectuar el análisis de la forma en que se ha desarrollado la ejecución concurrente de los hilos y la sincronización de los mismos, que es el verdadero objetivo, más allá de la utilización como ejemplo del problema de productores y consumidores.

6.3. Descripción del Algoritmo Utilizado

Los principales aspectos del algoritmo desarrollado son los siguientes:

- Los procesos productores simulan generar, es decir *grabar* información en un grupo de buffers disponibles.
- Los procesos consumidores simulan retirar, es decir *leer* información del grupo de buffer disponibles, que previamente debió ser *cargado* por los procesos productores.
- Los procesos consumidores sólo pueden *leer* información previamente *grabada* por los procesos productores.
- Si cuando un proceso consumidor intenta leer no hay suficientes buffers disponibles, debe esperar a que alguno de los procesos productores grabe los buffers.
- Los procesos productores sólo actúan cuando el nivel de buffers grabados está por debajo de cierto límite, llamado límite de reposición, que es un dato variable que se ingresa como parámetro de configuración de la simulación, al igual que la cantidad de buffers grabados en cada reposición.
- Los procesos consumidores pueden retirar (leer) buffers en número variable pero hasta cierto límite, que también es un dato variable que se ingresa como parámetro de configuración de la simulación.
- Debido a que los procesos productores sólo actúan cuando el número de buffers disponibles está por debajo del nivel de reposición y a que la cantidad de reposición es finita, se asegura que el número de buffers del pool será siempre finito.
- El tiempo que dura la simulación también es un parámetro de configuración que se ingresa como dato expresado en milisegundos.

6.4. Applet Desarrollado

El applet desarrollado recibe como entradas por pantalla los datos de configuración de la simulación referidos a:

- Número inicial de buffers ocupados.
- Número máximo de buffers leídos por vez.
- Número mínimo de buffers ocupados antes de grabar más.
- Número de buffers grabados por vez.
- Número de milisegundos de la ejecución.

Respecto del arreglo generado, el mismo tendrá un tamaño variable según los datos que se hayan suministrado a la ejecución que lo produce, especialmente el tiempo de ejecución; es preciso señalar además que el arreglo se aloja en la memoria temporal, es decir que se borra al terminar cada ejecución.

El código del applet desarrollado es el siguiente:

```
class Almacen {  
  
    public int stock=0;  
  
    private boolean necesitareponer;  
  
    public boolean cerrado=false;  
  
    public int MAXCOMPRA=0;  
  
    public int LIMITEREPONER=0;  
  
    public int CANTIDADREPONER=0;  
  
    public long TIEMPOAPERTURA=0;  
  
    long hora;  
  
    HilosApplet ha;  
  
    public Almacen(HilosApplet ha){  
  
        stock = Integer.parseInt(HilosApplet.stock);  
  
        MAXCOMPRA = Integer.parseInt(HilosApplet.maxcompra);  
  
        LIMITEREPONER = Integer.parseInt(HilosApplet.limitereponer);  
  
        CANTIDADREPONER = Integer.parseInt(HilosApplet.cantidadreponer);  
  
    }  
  
}
```

```
TIEMPOAPERTURA = Integer.parseInt(HilosApplet.tiempoapertura);
hora=System.currentTimeMillis();
this.ha = ha;
}
public synchronized void comprar(int consumidor, int cantidad) {
while (necesitareponer == true) {
try {
this.wait();
} catch (InterruptedException e) {
}}
if(!cerrado){
if(stock<cantidad){
cantidad=stock;
}
stock-=cantidad;
String c;
c = "Quedan "+ stock + unidades. El Consumidor "+ consumidor +
"ha leído "+cantidad+unidades.";
this.salida(c);
if(stock <= LIMITEREPONER){
necesitareponer = true;
this.notify();
}}
}
```

```
public synchronized void reponer(int productor) {
    if (!cerrado){
        while (necesitareponer == false) {
            try {
                this.wait();
            } catch (InterruptedException e) {}
        }
        stock+=CANTIDADREPONER;
        String c;
        c = "Quedan "+ stock + unidades. El Productor "+ productor +
            "ha grabado "+ CANTIDADREPONER + unidades.";
        this.salida(c);
        necesitareponer = false;
    }
    this.notifyAll();
}

public void salida(String texto){
    ha.salida(texto);
}

class Consumidor extends Thread {
    private Almacen almacen;
    private int numero;
    public Consumidor(Almacen a, int numero) {
        almacen = a;
```

```
this.numero = numero;
}
public void run() {
int value;
while(!almacen.cerrado){
value=(int)(Math.random()*almacen.MAXCOMPRA);
almacen.comprar(numero,value);
}}
}
public class HilosApplet extends javax.swing.JApplet {
public void init() {
try {
for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
if ("Nimbus".equals(info.getName())) {
javax.swing.UIManager.setLookAndFeel(info.getClassName());
break;}}
} catch (ClassNotFoundException ex) {
java.util.logging.Logger.getLogger(HilosApplet.class.getName())
.log(java.util.logging.Level.SEVERE, null, ex);
} catch (InstantiationException ex) {
java.util.logging.Logger.getLogger(HilosApplet.class.getName())
.log(java.util.logging.Level.SEVERE, null, ex);
} catch (IllegalAccessException ex) {
```

```
java.util.logging.Logger.getLogger(HilosApplet.class.getName())
.log(java.util.logging.Level.SEVERE, null, ex);
} catch (javax.swing.UnsupportedLookAndFeelException ex) {
java.util.logging.Logger.getLogger(HilosApplet.class.getName())
.log(java.util.logging.Level.SEVERE, null, ex);
}try {
java.awt.EventQueue.invokeAndWait(new Runnable() {
public void run() {
initComponents();
getDatos();
}});
} catch (Exception ex) {
ex.printStackTrace();
}}
private void getDatos(){
stock = this.jTextField1.getText() ;
maxcompra = this.jTextField2.getText();
limitereponer = this.jTextField3.getText();
cantidadreponer = this.jTextField4.getText();
tiempoapertura = this.jTextField5.getText();
}
public void salida(String text){
jTextArea1.append(text + "\n");
}
```

```
private void initComponents() {  
jTabbedPane1 = new javax.swing.JTabbedPane();  
jPanel1 = new javax.swing.JPanel();  
jScrollPane1 = new javax.swing.JScrollPane();  
jTextArea1 = new javax.swing.JTextArea();  
jButton1 = new javax.swing.JButton();  
jPanel2 = new javax.swing.JPanel();  
jLabel1 = new javax.swing.JLabel();  
jLabel2 = new javax.swing.JLabel();  
jLabel3 = new javax.swing.JLabel();  
jLabel4 = new javax.swing.JLabel();  
jTextField1 = new javax.swing.JTextField();  
jTextField2 = new javax.swing.JTextField();  
jTextField3 = new javax.swing.JTextField();  
jTextField4 = new javax.swing.JTextField();  
jButton2 = new javax.swing.JButton();  
jLabel5 = new javax.swing.JLabel();  
jTextField5 = new javax.swing.JTextField();  
jTextArea1.setColumns(20);  
jTextArea1.setEditable(false);  
jTextArea1.setRows(5);  
jScrollPane1.setViewportView(jTextArea1);  
jButton1.setText("Ejecutar");  
jButton1.addActionListener(new java.awt.event.ActionListener() {
```

```
public void actionPerformed(java.awt.event.ActionEvent evt) {
    jButton1ActionPerformed(evt);
}});

javax.swing.GroupLayout jPanel1Layout = new javax.swing
    .GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout
        .Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
            jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane1, javax.swing.GroupLayout
                    .DEFAULT_SIZE, 463, Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jButton1)));
jPanel1Layout.setVerticalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout
        .Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addContainerGap()
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addContainerGap()
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addContainerGap()
                                .addComponent(jButton1, javax.swing.GroupLayout
```

```
.Alignment.TRAILING)
.addComponent(jScrollPane1, javax.swing.GroupLayout
.DEFAULT_SIZE, 265, Short.MAX_VALUE))
.addContainerGap());
jTabbedPane1.addTab(Resultados", jPanel1);
jLabel1.setText("Número inicial de buffers ocupados:");
jLabel2.setText("Número máximo de buffers leídos por vez:");
jLabel3.setText("Número mínimo de buffers ocupados antes de grabar más:");
jLabel4.setText("Número de buffers ocupados antes de grabar más:");
jTextField1.setColumns(4);
jTextField1.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField1.setText("0");
jTextField2.setColumns(4);
jTextField2.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField2.setText("30");
jTextField3.setColumns(4);
jTextField3.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField3.setText("40");
jTextField4.setColumns(4);
jTextField4.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField4.setText("50");
jButton2.setText(Configure");
jButton2.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
jButton2ActionPerformed(evt);
    });
jLabel5.setText("Número de milisegundos de la ejecución:");
jTextField5.setColumns(4);
jTextField5.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextField5.setText("2000");
javax.swing.GroupLayout jPanel2Layout = new javax.swing
    .GroupLayout(jPanel2);
jPanel2.setLayout(jPanel2Layout);
jPanel2Layout.setHorizontalGroup(
    jPanel2Layout.createParallelGroup(javax.swing
    .GroupLayout.Alignment.LEADING)
    .addGroup(jPanel2Layout.createSequentialGroup()
    .addGap(26, 26, 26)
    .addGroup(jPanel2Layout.createParallelGroup(javax.swing
    .GroupLayout.Alignment.TRAILING)
    .addComponent(jButton2)
    .addGroup(jPanel2Layout.createParallelGroup(javax.swing
    .GroupLayout.Alignment.LEADING)
    .addComponent(jLabel5)
    .addComponent(jLabel1)
    .addComponent(jLabel3)
    .addComponent(jLabel2)
    .addComponent(jLabel4)))
```

```
.addPreferredGap(javax.swing.LayoutStyle  
.ComponentPlacement.RELATED, 170, Short.MAX_VALUE)  
.addGroup(jPanel2Layout.createParallelGroup(javax.swing  
.GroupLayout.Alignment.LEADING)  
.addComponent(jTextField4, javax.swing.GroupLayout  
.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,  
javax.swing.GroupLayout.PREFERRED_SIZE)  
.addComponent(jTextField1, javax.swing  
.GroupLayout.PREFERRED_SIZE, javax.swing  
.GroupLayout.DEFAULT_SIZE, javax.swing  
.GroupLayout.PREFERRED_SIZE)  
.addComponent(jTextField2, javax.swing  
.GroupLayout.PREFERRED_SIZE, javax.swing  
.GroupLayout.DEFAULT_SIZE, javax.swing  
.GroupLayout.PREFERRED_SIZE)  
.addComponent(jTextField3, javax.swing  
.GroupLayout.PREFERRED_SIZE, javax.swing  
.GroupLayout.DEFAULT_SIZE, javax.swing  
.GroupLayout.PREFERRED_SIZE)  
.addComponent(jTextField5, javax.swing  
.GroupLayout.PREFERRED_SIZE, javax.swing  
.GroupLayout.DEFAULT_SIZE, javax.swing  
.GroupLayout.PREFERRED_SIZE))  
.addGap(38, 38, 38));
```

```
jPanel2Layout.setVerticalGroup(  
jPanel2Layout.createParallelGroup(javax.swing  
.GroupLayout.Alignment.LEADING)  
.addGroup(jPanel2Layout.createSequentialGroup()  
.addGap(26, 26, 26)  
.addGroup(jPanel2Layout.createParallelGroup  
(javax.swing.GroupLayout.Alignment.LEADING)  
.addGroup(jPanel2Layout.createSequentialGroup()  
.addComponent(jLabel1).addGap(18, 18, 18)  
.addComponent(jLabel2).addGap(18, 18, 18)  
.addGroup(jPanel2Layout.createParallelGroup  
(javax.swing.GroupLayout.Alignment.BASELINE)  
.addComponent(jLabel3)  
.addComponent(jTextField3, javax.swing  
.GroupLayout.PREFERRED_SIZE, javax.swing  
.GroupLayout.DEFAULT_SIZE, javax.swing  
.GroupLayout.PREFERRED_SIZE)))  
.addGroup(jPanel2Layout.createSequentialGroup()  
.addComponent(jTextField1, javax.swing  
.GroupLayout.PREFERRED_SIZE, javax.swing  
.GroupLayout.DEFAULT_SIZE, javax.swing  
.GroupLayout.PREFERRED_SIZE)  
.addPreferredGap(javax.swing.LayoutStyle  
.ComponentPlacement.RELATED)
```

```
.addComponent(jTextField2, javax.swing
 GroupLayout.PREFERRED_SIZE, javax.swing
 GroupLayout.DEFAULT_SIZE, javax.swing
 GroupLayout.PREFERRED_SIZE)))
.addGap(18, 18, 18)
.addGroup(jPanel2Layout.createParallelGroup
 (javax.swing.GroupLayout.Alignment.LEADING)
 .addComponent(jTextField4, javax.swing
 GroupLayout.PREFERRED_SIZE, javax.swing
 GroupLayout.DEFAULT_SIZE, javax.swing
 GroupLayout.PREFERRED_SIZE)
 .addComponent(jLabel4)).addGap(18, 18, 18)
.addGroup(jPanel2Layout.createParallelGroup
 (javax.swing.GroupLayout.Alignment.LEADING)
 .addComponent(jLabel5)
 .addGroup(jPanel2Layout.createSequentialGroup())
 .addComponent(jTextField5, javax.swing.GroupLayout.
 PREFERRED_SIZE, javax.swing.GroupLayout
 .DEFAULT_SIZE, javax.swing
 GroupLayout.PREFERRED_SIZE).addGap(11, 11, 11)
 .addComponent(jButton2)).addContainerGap());
jTabbedPane1.addTab("Configuración", jPanel2);
javax.swing.GroupLayout layout = new javax.swing
 GroupLayout(getContentPane());
```

```
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addContainerGap()
.addComponent(jTabbedPane1, javax.swing
.GroupLayout.PREFERRED_SIZE, 557, javax.swing
.GroupLayout.PREFERRED_SIZE)
.addContainerGap(31, Short.MAX_VALUE)));
layout.setVerticalGroup(
layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addContainerGap()
.addComponent(jTabbedPane1, javax.swing
.GroupLayout.PREFERRED_SIZE, 315, javax.swing
.GroupLayout.PREFERRED_SIZE)
.addContainerGap(26, Short.MAX_VALUE)));
}
private void jButton1ActionPerformed
(java.awt.event.ActionEvent evt) {
this.jTextArea1.setText();
try {
```

```
synchronized(this){
String d, asterisc;
asterisc = "\n*****";
d = "\n ";
this.jTextArea1.append(asterisc);
this.jTextArea1.append
("\nInicio de la ejecución concurrente con hilos.");
this.jTextArea1.append
("\nLos datos de configuración son los siguientes: ");
this.jTextArea1.append
("\nNúmero inicial de buffers ocupados: "+ stock + ".");
this.jTextArea1.append
("\nNúmero máximo de buffers leídos por vez: "+ limitereponer + ".");
this.jTextArea1.append
("\nNúmero mínimo de buffers ocupados antes de grabar más: "+ maxcompra +
".");
this.jTextArea1.append
("\nNúmero de buffers grabados por vez: "+ cantidadreponer + ".");
this.jTextArea1.append
("\nNúmero de milisegundos de la ejecución: "+ tiempoapertura + ".");
this.jTextArea1.append(asterisc); this.jTextArea1.append(d);
}
Almacen a = new Almacen(this);
Productor p1 = new Productor(a,1);
Productor p2 = new Productor(a,2);
```

```
Consumidor c1 = new Consumidor(a,1);
Consumidor c2 = new Consumidor(a,2);
Consumidor c3 = new Consumidor(a,3);
p1.start();
p2.start();
c1.start();
c2.start();
c3.start();
} catch (java.lang.Exception ex) {
// Atrapa excepciones y las despliega.
ex.printStackTrace ();
} }
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
this.getDatos();
}
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
```

```
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTabbedPane jTabbedPane1;
private javax.swing.JTextArea jTextArea1;
private javax.swing.JTextField jTextField1;
private javax.swing.JTextField jTextField2;
private javax.swing.JTextField jTextField3;
private javax.swing.JTextField jTextField4;
private javax.swing.JTextField jTextField5;
public static String salida;
public static String stock;
public static String maxcompra;
public static String limitereponer;
public static String cantidadreponer;
public static String tiempoapertura;
}
class Productor extends Thread {
private Almacen almacen;
private int numero;
public Productor(Almacen a, int numero) {
almacen = a;
this.numero = numero;
}
public void run() {
if(almacen.cerrado) System.out.print("Cerro");
```

```
while(!almacen.cerrado){
if((System.currentTimeMillis() - almacen.hora) > almacen
.TIEMPOAPERTURA){
almacen.cerrado=true;
String c;
c = "Fin de la ejecución.";
almacen.salida(c);
}else{
almacen.reponer(numero);
try {
sleep((int)(Math.random() * 100));
}
catch (InterruptedException e) {
} } } }
}
```

6.5. Datos y Ejecuciones

El applet consta de dos solapas, en una de ellas se podrá configurar el número inicial de buffer ocupados, el número máximo de buffer leídos por vez, el número de buffer mínimos ocupados antes de grabar más, el número de buffer grabados por vez y el tiempo de ejecución de la simulación en milisegundos, valores necesarios para iniciar la simulación.

Luego de cargar los datos en la solapa de configuración, en la solapa de resultados se puede ejecutar la simulación para observar los valores detallados de la simulación que se muestran paso a paso en pantalla. Ver figura 6.1 de la página 122 y la figura 6.2 de la página 122.

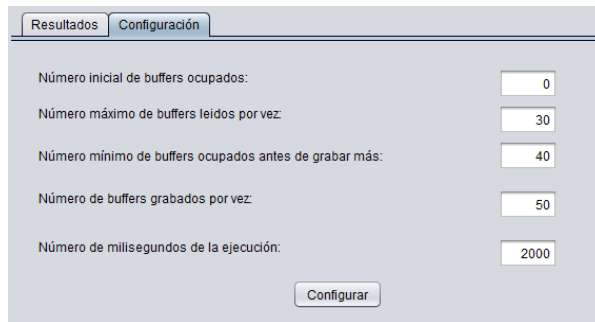


Figura 6.1: Pantalla de configuración

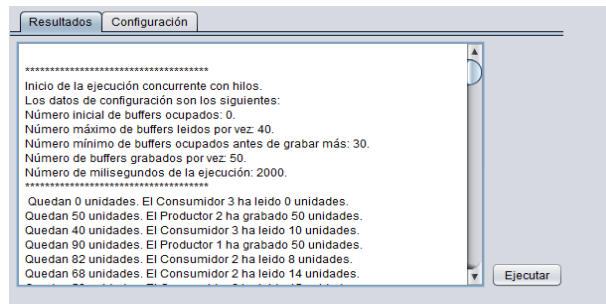


Figura 6.2: Pantalla de resultados

6.6. Resultados y Conclusiones

Los resultados obtenidos con el applet desarrollado son compatibles con lo previsto por la teoría en cuanto a las facilidades de los hilos o procesos ligeros para resolver problemas de concurrencia y de simultaneidad, en caso de disponer de multiprocesadores; esto los hace especialmente aplicables a un gran número de problemas propios de los sistemas operativos como administradores de recursos compartidos, siendo el caso del problema planteado un caso genérico de la problemática antes mencionada.

Capítulo 7

Sincronización de los Procesos en Sistemas Distribuidos – Exclusión Mutua

7.1. Introducción

Como los casos de estudio anteriores, este caso fue desarrollado en un applet, pero en esta ocasión se tiene como escenario un esquema de sistemas operativos distribuidos.

El desarrollo del algoritmo aplicado a la simulación de sincronización de los procesos en sistemas distribuidos, fue creado partiendo desde los conceptos teóricos, utilizando hilos para simular los procesos y arreglos para simular las regiones críticas, logrando de esta manera poder observar el comportamiento de los mismos ejecutando la simulación en un tiempo preestablecido.

7.2. Objetivo

Conforme a lo antes indicado, el objetivo fue el de codificar un programa (applet) en Java que efectuara la simulación del comportamiento de los procesos en sistemas operativos distribuidos al compartir recursos críticos en un entorno de estas características mediante la exclusión mutua, visualizando en

pantalla los datos obtenidos en un tiempo determinado por el usuario para su posterior análisis.

7.3. Introducción a la Sincronización en Sistemas Distribuidos

Cuando se habla de sistemas operativos distribuidos, se hace referencia a un conjunto de computadores autónomos que se encuentran físicamente en lugares distintos, comunicados entre sí, trabajando en conjunto, potenciando la capacidad de procesamiento y dando al usuario la percepción de un sistema operativo único e integrado.

La comunicación entre estos computadores se realiza por medio del envío de mensajes entre ellos, a través de un protocolo prefijado por un esquema cliente-servidor.

Pero además de la comunicación, es fundamental la forma en que los procesos:

- Cooperan.
- Se sincronizan entre sí.

Ejemplos:

- La forma de implantar las regiones críticas.
- La forma de asignar recursos en un sistema distribuido.

Los problemas relativos a las regiones críticas, exclusión mutua y la sincronización:

- Generalmente se resuelven en sistemas de un solo procesador, con métodos como los semáforos y los monitores:
 - se basan en la memoria compartida:
 - no son aplicables a sistemas distribuidos. [1] [2]

Otro problema de gran importancia es el tiempo y la forma de medirlo, que juega un papel fundamental en algunos modelos de sincronización.

Generalmente los algoritmos distribuidos tienen las siguientes propiedades:

- La información relevante se distribuye entre varias máquinas.
- Los procesos toman las decisiones solo con base en la información disponible en forma local.
- Debe evitarse un único punto de fallo en el sistema.
- No existe un reloj común o alguna otra fuente precisa del tiempo global.

Los primeros tres puntos indican que es inaceptable reunir toda la información en un solo lugar para su procesamiento; lograr la sincronización sin centralización requiere hacer las cosas distintas al caso de los sistemas operativos tradicionales.

El último punto también es crucial:

- En un sistema centralizado el tiempo no es ambiguo.
- En un sistema distribuido no es trivial poner de acuerdo a todas las máquinas en la hora.
- Se requiere un acuerdo global en el tiempo; la falta de sincronización en los relojes puede ser drástica en procesos dependientes del tiempo.

7.4. Exclusión Mutua

Cuando un proceso debe leer o actualizar ciertas estructuras de datos compartidas, primero debe ingresar a una región crítica para lograr la exclusión mutua y garantizar que ningún otro proceso utilizará las estructuras de datos al mismo tiempo.

En sistemas monoprocesadores las regiones críticas se protegen con semáforos, monitores y similares.

En sistemas distribuidos la cuestión es más compleja.

7.5. Un Algoritmo Centralizado

La forma más directa de lograr la exclusión mutua en un sistema distribuido es simular a la forma en que se lleva a cabo en un sistema monoprocesador.

Se elige un proceso coordinador.

Cuando un proceso desea ingresar a una región crítica:

- Envía un mensaje de solicitud al coordinador:
 - Indicando la región crítica.
 - Solicitando permiso de acceso.
- Si ningún otro proceso está en ese momento en esa región crítica:
 - El coordinador envía una respuesta otorgando permiso.
- Cuando llega la respuesta el proceso solicitante entra a la región crítica.

Si un proceso pide permiso para entrar a una región ya asignada a otro proceso:

- El coordinador no otorga el permiso y encola el pedido.

Cuando un proceso sale de la región crítica envía un mensaje al coordinador para liberar su acceso exclusivo:

- El coordinador extrae el primer elemento de la cola de solicitudes diferidas y envía a ese proceso un mensaje otorgando el permiso:
 - El proceso queda habilitado para acceder a la región crítica solicitada.

Es un esquema sencillo, justo y con pocos mensajes de control.

La limitante es que el coordinador puede ser un cuello de botella, puede fallar y bloquear a los procesos que esperan una respuesta de habilitación de acceso. [1] [4]

7.6. Applet Desarrollado

El applet que se desarrolló consiste en un conjunto de hilos que representan los procesos distribuidos, los que tienen un coordinador, un tiempo de espera antes de iniciarse, un tiempo de uso de las distintas regiones críticas, también el porcentaje de los mismos que terminarán de forma incorrecta. En el transcurso de la simulación, estos procesos competirán por las regiones críticas que están representadas en el applet por un arreglo dimensionado por el usuario.

El código del programa desarrollado es el siguiente:

```
public class Proceso implements Runnable {
    private int _idProceso;
    private Proceso Coordinador;
    int esperaInicial; // es el tiempo que va a esperar el
proceso antes de iniciar la ejecucion
    boolean enEjecucion; // para controlar la ejecucion del hilo
    Thread t; // el hilo
    private boolean muriendo;
    public void solicitarRegion(Proceso p, int idReg) {
        if (Simulador.regiones[idReg].enUso()) {
            p.suspend();
            encolar(p, idReg, 1);
        } else {
            SimuladorApplet.sim.mostrar("Se concede la región " + idReg + " al Proceso "
+ p);
            Simulador.regiones[idReg].ocupar(p.idProceso());
        }
        //si esta libre la region la marcamos como ocupada
    }
}
```

```

//si no ponemos a dormir el proceso y la encolamos
}

void encolar(Proceso pro, int idReg, int modo){
pro.suspender();
if (modo ==1){
int[] cola = new int[2];
cola[0]=pro.idProceso();
cola[1]=idReg;
SimuladorApplet.sim.mostrar("Se encola proceso: "+pro);
Simulador.colaDeProcesos.add(cola);
}else if(modo ==2){
//se encola en las colas por regiones
}else{
//igual que modo 2 pero tien en cuenta las prioridades
} }

public void desencolar(int idReg, int modo){
if (modo ==1){
int aux[]=new int[2];
boolean encontrado = false;
for(int i =0; i<Simulador.colaDeProcesos.size()-1; i++){
aux=(int[])Simulador.colaDeProcesos.get(i);
if (idReg==aux[1]){
encontrado = true;

SimuladorApplet.sim.mostrar("Se desencola el proceso " + (char) ('A' + aux[0])
+ " y se concede la región " + idReg);

```

```
Simulador.colaDeProcesos.remove(i);
Simulador.procesos[aux[0]].reanudar();
Simulador.regiones[idReg].ocupar(aux[0]);
}
if (encontrado){ break;}
}}
public void liberarRegion(int idReg){
//marca como libre la region y desencola un proceso de esa region
Simulador.regiones[idReg].liberar();
this.desencolar(idReg, 1);
this.controlarProcesos();
}
public void run() {
if (!this.esCoord()) {
try {
Thread.sleep(this.esperaInicial);
} catch (InterruptedException ie) {
if (this.muriendo) {
return;
} }
}
SimuladorApplet.sim.mostrar("Iniciando Proceso " + this);
while (t != null) {
if (this.enEjecucion) {
```

```
if (this.esCoord()) {
    this.controlarProcesos();
    Thread.yield();
    //espera si es coordinador
} else {
    int reg = this.buscoRegion();
    // Si no está en ejecución, es porque está suspendido
    // entonces sale del bucle
    if (!this.enEjecucion) {
        continue;
    };
    // Usará la región por un un tiempo de hasta el 30 % de duración de la simulación.
    int tiempoDeUso = (int) (Math.random() * Simulador.tduracion * 0.3);
    try {
        Thread.sleep(tiempoDeUso);
    } catch (InterruptedException ie) {
        if (this.muriendo) {
            return;
        }
    }
    if (this.muriendo) {
        this.terminar();
        SimuladorApplet.sim.mostrar("El proceso " + this + " está muriendo...");
    }
    return;
}
```

```
SimuladorApplet.sim.mostrar(" El proceso " + this + " libera la región " + reg);
this.Coordinador.liberarRegion(reg);

//calcula la region y cuanto tiempo de uso aleatoriamente
//pide la region al coordinador
//duerme por el tiempo de uso
//avisa al coordinador que libera la region
}

// Main.sim.mostrar(this.toString());
} }

if (!this.muriendo) {
SimuladorApplet.sim.mostrar(" El Proceso" + this + "ha finalizado.");
}

t = null;
}

public Proceso(int id, Proceso coord, int tie, boolean seMuere){
this.esperaInicial = (int)(Math.random() * tie * (200));
this.Coordinador=coord;
this._idProceso=id;
this.enEjecucion = true;
this.seMuere = seMuere;
t = new Thread(this);
}

public void iniciar(){
t.start();
```

```
    }  
    public void suspender(){  
        this.enEjecucion = false;  
    }  
    public void reanudar(){  
        this.enEjecucion = true;  
    }  
    public void terminar(){  
        this.enEjecucion = false;  
        t = null;  
    }  
    public void idProceso(int idProc){  
        this._idProceso=idProc;  
    }  
    public boolean esCoord(){  
        return this.Coordinador==null;  
    }  
    public int idProceso(){  
        return this._idProceso;  
    }  
    public String toString(){  
        String s=new String();  
        s = " " +(char) ('A'+ this._idProceso);  
        if (this.esCoord()){
```

```
s+=Coordinador ";
}
return s;
}
public void matar(){
    SimuladorApplet.sim.mostrar("El proceso " + this + " termina de manera in-
esperada.");
    this.muriendo = true;
    t.interrupt();
}
private int buscoRegion() {
    int reg = -1;
    for(int i = 0; i < Simulador.regiones.length ; i++){
        if (Simulador.regiones[i].ocupandoRegion(this._idProceso)){
            reg = Simulador.regiones[i].idRegion();
        }
    }
    if (reg == -1){
        reg = (int) (Math.random() * (Simulador.regiones.length - 1));
        SimuladorApplet.sim.mostrar(" El proceso " + this + " solicita la región " +
reg);
        this.Coordinador.solicitarRegion(this, reg);
    }
    return reg;
}
```

```

private void controlarProcesos() {
// Recorre las regiones para verificar que
// los procesos que las ocupan siguen vivos

for(Region r : Simulador.regiones){
for(Proceso p : Simulador.procesos){
if (p.muriendo && r.ocupandoRegion(p.idProceso())){
SimuladorApplet.sim.mostrar("-----");
SimuladorApplet.sim.mostrar("El proceso " + p + " ya no se está ejecutando");
SimuladorApplet.sim.mostrar("y mantiene bloqueada la región\n" + r + ".");
SimuladorApplet.sim.mostrar("Se fuerza la liberación de dicha región.");
this.liberarRegion(r.idRegion());
SimuladorApplet.sim.mostrar("-----");
} }
} }
}

public class Region {
private int _idRegion;
private boolean _enUso;
private int _idProceso; // El proceso que tenga asignado esta región;
public Region(int idReg){
this._idRegion=idReg;
this._enUso=false;
this._idProceso = -1;
}
}

```

```
    }  
    public void liberar(){  
        SimuladorApplet.sim.mostrar("Se libero la región "+this.idRegion());  
        this._enUso=false;  
        this._idProceso = -1;  
    }  
    public void ocupar(int idProceso){  
        this._idProceso = idProceso;  
        this._enUso=true;  
    }  
    public boolean enUso (){  
        return this._enUso;  
    }  
    public void idRegion(int idReg){  
        this._idRegion=idReg;  
        this._enUso= false;  
    }  
    public int idRegion(){  
        return this._idRegion;  
    }  
    public String toString(){  
        String s=new String();  
        s = "Id Región: "+this._idRegion+ " Estado de uso: " + this._enUso;  
        return s;  
    }
```

```
    }  
    public boolean ocupandoRegion(int idProceso){  
        return this._idProceso == idProceso;  
    }  
}  
  
public class Simulador extends Thread {  
    public static ArrayList colaDeProcesos;  
    public static Region regiones[];  
    public static Proceso procesos[];  
    public static int tduracion;  
    private int cantPro;  
    private int cantReg;  
    private int tiempo;  
    private JTextArea salida;  
    private java.util.Timer timer;  
    Simulador(int cantPro, int cantReg, int tiempo, JTextArea salida){  
        this.salida = salida;  
        this.timer = new java.util.Timer();  
        Simulador.tduracion = tiempo * 1000;  
        this.cantPro=cantPro;  
        this.cantReg=cantReg;  
        this.tiempo=tiempo;  
    }  
    public void run(){
```

```
regiones = new Region[cantReg];
colaDeProcesos = new ArrayList(cantPro);
procesos = new Proceso[cantPro];
for (int i = 0; i < regiones.length; i++) { //se crean las regiones
regiones[i] = new Region(i);
mostrar(regiones[i].toString());
}
//Se genera un array con los id de los procesos que han de morir antes de terminar.
Random r = new Random();
int[] seMueren = new int[(int) (procesos.length * 0.2)];
for (int i = 0; i < seMueren.length; i++) {
seMueren[i] = r.nextInt(cantPro);
}
int coor = (int) (Math.random() * procesos.length); //calcula el id del coordi-
nador
procesos[coor] = new Proceso(coor, null, tiempo, false);
boolean morir;
for (int i = 0; i < procesos.length; i++) { //se crean las regiones
morir = false;
for (int j = 0; j < seMueren.length; j++) {
//System.out.println(j);
if (seMueren[j] == i) {
morir = true;
} }
if (i != coor) {
```

```
procesos[i] = new Proceso(i, procesos[coor], tiempo, morir);
if (morir) {
agregarTerminarProceso(new TerminarProceso(procesos[i], Simulador.tduracion));
}
}
mostrar(procesos[i].toString());
}
for (Proceso p : procesos) {
p.iniciar();
}
try {
Thread.sleep(Simulador.tduracion);
} catch (InterruptedException ie) {
}
for (int i = 0; i < procesos.length; i++) {
procesos[i].terminar();
} }
public void mostrar(String s){
this.salida.append(s+"\n");
}
public void agregarTerminarProceso(TerminarProceso t){
this.timer.schedule(t, t.tiempo());
} }
public class SimuladorApplet extends JApplet {
```

```
public void init() {
    try {
        java.awt.EventQueue.invokeAndWait(new Runnable() {
            public void run() {
                initComponents();
            }
        });
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

private void initComponents() {
    jPanel1 = new javax.swing.JPanel();
    jScrollPane1 = new javax.swing.JScrollPane();
    jTextArea1 = new javax.swing.JTextArea();
    jPanel2 = new javax.swing.JPanel();
    jLabel1 = new javax.swing.JLabel();
    jTextField1 = new javax.swing.JTextField();
    jLabel2 = new javax.swing.JLabel();
    jTextField2 = new javax.swing.JTextField();
    jTextField3 = new javax.swing.JTextField();
    jLabel3 = new javax.swing.JLabel();
    jButton1 = new javax.swing.JButton();
    panelDibujo = new PanelDibujo(this.sim.regiones);
    javax.swing.GroupLayout jPanel1Layout = new javax.swing
    .GroupLayout(jPanel1);
```

```
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
jPanel1Layout.createParallelGroup(javax.swing
 GroupLayout.Alignment.LEADING)
.addGap(0, 100, Short.MAX_VALUE) );
jPanel1Layout.setVerticalGroup(
jPanel1Layout.createParallelGroup(javax.swing
 GroupLayout.Alignment.LEADING)
.addGap(0, 100, Short.MAX_VALUE) );
jTextArea1.setColumns(20);
jTextArea1.setEditable(false);
jTextArea1.setRows(5);
jTextArea1.setName("salida");
jScrollPane1.setViewportView(jTextArea1);
jPanel2.setName("panelIngresos");
jLabel1.setText("Cant. Procesos");
jTextField1.setText("10");
jLabel2.setText("Cant. Regiones Criticas");
jTextField2.setText("5");
jTextField3.setText("10");
jLabel3.setText("Tiempo de Ejecucion");
jButton1.setText("Comenzar");
jButton1.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
jButton1ActionPerformed(evt); } });  
javax.swing.GroupLayout jPanel2Layout = new javax  
.swing.GroupLayout(jPanel2);  
jPanel2.setLayout(jPanel2Layout);  
jPanel2Layout.setHorizontalGroup(  
jPanel2Layout.createParallelGroup(javax.swing  
.GroupLayout.Alignment.LEADING)  
.addGroup(jPanel2Layout.createSequentialGroup()  
.addContainerGap()  
.addGroup(jPanel2Layout.createParallelGroup(javax.swing  
.GroupLayout.Alignment.TRAILING)  
.addComponent(jLabel1)  
.addComponent(jLabel2)  
.addComponent(jLabel3))  
.addPreferredGap(new javax.swing.LayoutStyle  
.ComponentPlacement.RELATED)  
.addGroup(jPanel2Layout.createParallelGroup(javax.swing  
(javax.swing.GroupLayout.Alignment.LEADING, false)  
.addComponent(jTextField2, javax.swing  
.GroupLayout.Alignment.TRAILING)  
.addComponent(jTextField3, javax.swing  
.GroupLayout.Alignment.TRAILING)  
.addComponent(jTextField1, javax.swing  
.GroupLayout.Alignment.TRAILING, javax.swing
```

```
.GroupLayout.PREFERRED_SIZE, 47, javax.swing
.GroupLayout.PREFERRED_SIZE))
.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel2Layout.createSequentialGroup())
.addContainerGap(92, Short.MAX_VALUE)
.addComponent(jButton1)
.addContainerGap());
jPanel2Layout.setVerticalGroup( jPanel2Layout
.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(jPanel2Layout.createSequentialGroup())
.addContainerGap()
.addGroup(jPanel2Layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.TRAILING)
.addGroup(jPanel2Layout.createSequentialGroup())
.addPreferredGap(javax.swing.LayoutStyle
ComponentPlacement.RELATED, 6, javax.swing
.GroupLayout.PREFERRED_SIZE)
.addGroup(jPanel2Layout.createParallelGroup(javax.swing
.GroupLayout.Alignment.LEADING)
.addGroup(jPanel2Layout.createSequentialGroup())
.addComponent(jLabel1)
.addGap(26, 26, 26)
```

```
.addComponent(jLabel2))
.addGroup(jPanel2Layout.createSequentialGroup())
.addGap(84, 84, 84)
.addComponent(jLabel3))))
.addGroup(jPanel2Layout.createSequentialGroup())
.addComponent(jTextField1, javax.swing.GroupLayout
.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addGap(26, 26, 26)
.addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing
.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle
.ComponentPlacement.RELATED, 18, Short.MAX_VALUE)
.addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing
.GroupLayout.PREFERRED_SIZE)))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(jButton1)
.addContainerGap());
javax.swing.GroupLayout panelDibujoLayout = new javax.swing
.GroupLayout(panelDibujo);
panelDibujo.setLayout(panelDibujoLayout);
panelDibujoLayout.setHorizontalGroup(
```

```
panelDibujoLayout.createParallelGroup(javax.swing
 GroupLayout.Alignment.LEADING)
.addGap(0, 220, Short.MAX_VALUE) );
panelDibujoLayout.setVerticalGroup(
panelDibujoLayout.createParallelGroup(javax.swing
 GroupLayout.Alignment.LEADING)
.addGap(0, 118, Short.MAX_VALUE) );
javax.swing.GroupLayout layout = new javax.swing
 GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
layout.createParallelGroup(javax.swing.GroupLayout
 Alignment.LEADING)
.addGroup(javax.swing.GroupLayout.Alignment
 TRAILING, layout.createSequentialGroup()
.addGap(106, 106, 106)
.addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE,
420, Short.MAX_VALUE)
.addPreferredGap(javax.swing.LayoutStyle
 ComponentPlacement.UNRELATED)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout
 Alignment.LEADING)
.addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing
```

```
.GroupLayout.PREFERRED_SIZE)
.addComponent(panelDibujo, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing
.GroupLayout.PREFERRED_SIZE))
.addContainerGap() );
layout.setVerticalGroup( layout.createParallelGroup(javax.swing
 GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addContainerGap()
.addGroup(layout.createParallelGroup(javax.swing
 GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing
 GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(panelDibujo, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
.addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.TRAILING
, javax.swing.GroupLayout.DEFAULT_SIZE, 279, Short.MAX_VALUE))
.addGap(30, 30, 30)) );
}
public void setJTextArea1(javax.swing.JTextArea area){
this.jTextArea1=area;
```

```
    }  
  
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
        if (sim == null || !sim.isAlive()){  
            int p1=Integer.parseInt(this.jTextField1.getText());  
            int p2=Integer.parseInt(this.jTextField2.getText());  
            int p3=Integer.parseInt(this.jTextField3.getText());  
            sim = new Simulador(p1, p2, p3, this.jTextArea1);  
            sim.start();  
        }else{  
        } }  
  
        private javax.swing.JButton jButton1;  
        private javax.swing.JLabel jLabel1;  
        private javax.swing.JLabel jLabel2;  
        private javax.swing.JLabel jLabel3;  
        private javax.swing.JPanel jPanel1;  
        private javax.swing.JPanel jPanel2;  
        private javax.swing.JScrollPane jScrollPane1;  
        private javax.swing.JTextArea jTextArea1;  
        private javax.swing.JTextField jTextField1;  
        private javax.swing.JTextField jTextField2;  
        private javax.swing.JTextField jTextField3;  
        private javax.swing.JPanel panelDibujo;  
        public static Simulador sim;  
    }
```

```
public class TerminarProceso extends TimerTask{
    private Proceso p;
    private long tiempo;
    TerminarProceso() throws Exception{
        throw new Exception("No se puede utilizar el Constructor predeterminado...");
    }
    TerminarProceso(Proceso p, long tiempo){
        this.p = p;
        this.tiempo = (p.esperaInicial)+(long)(Math.random() * tiempo * 0.2);
    }
    public void run(){
        p.matar();
    }
    public long tiempo(){
        return this.tiempo;
    }
}
```

7.7. Datos y Ejecuciones

Los datos para iniciar la simulación están predeterminados, pudiendo éstos ser modificados por el usuario, los valores iniciales son: la cantidad de procesos, la cantidad de regiones críticas y el tiempo de ejecución de la simulación en segundos.

El resultado de la interacción entre los procesos, la asignación, liberación y desbloques de las regiones críticas pueden ser observadas en la pantalla principal del applet para su posterior análisis, como se puede ver en la figura 7.1 de la página 149 y la figura 7.2 de la página 149.

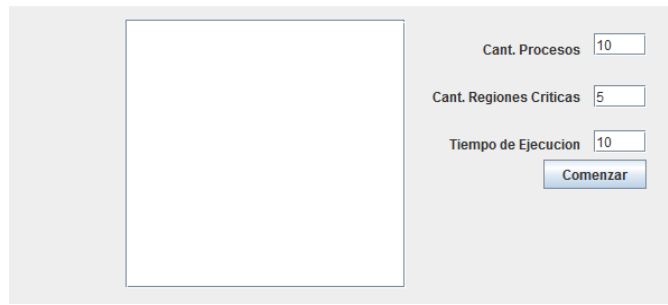


Figura 7.1: Pantalla Principal

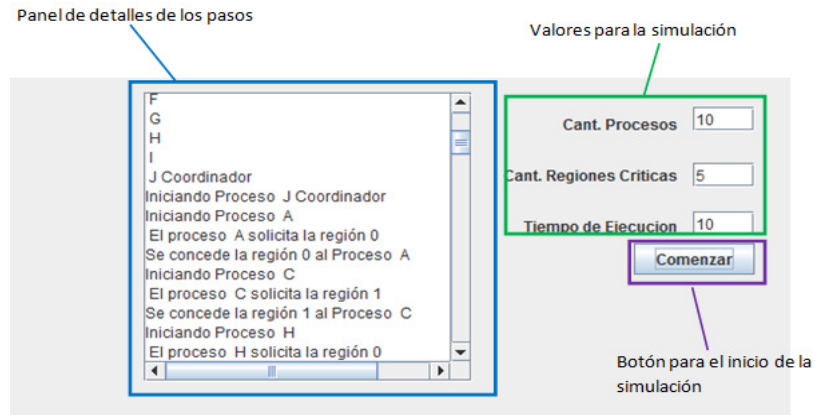


Figura 7.2: Resultados Obtenidos

7.8. Resultados y Conclusiones

Los resultados observados en la simulación cumplen con los objetivos propuestos, mostrando la interacción entre los procesos creados, compitiendo por las distintas regiones críticas, teniendo un proceso coordinador que actúa como administrador de las regiones críticas solicitadas por los demás procesos y desbloqueando las regiones críticas que fueron apropiadas por algún proceso que terminó de forma inesperada o errónea.

Este es un esquema sencillo y de fácil comprensión en lo que respecta a la sincronización de procesos, pero tiene como principal desventaja el hecho de ser un algoritmo centralizado, es decir, si termina de forma inesperada el coordinador, el esquema de sincronización deja de funcionar o cae, en este caso se debería implementar un sistema para la selección de un nuevo coordinador que se ponga en ejecución cuando se detecten problemas de este tipo.

Capítulo 8

La Aplicación

Este capítulo describe los componentes de la aplicación web desarrollada para contener a los applets que simulan cada uno de los algoritmos de administración de recursos de sistemas operativos desarrollados a lo largo de este trabajo.

El entorno web consta de un menú horizontal donde se encuentran los siguientes botones:

- Principal: muestra la página principal del sitio web.
- Ayuda: muestra la interfaz gráfica de cada applet, indicando en cada caso los sectores más importantes del mismo.
- Acerca de: muestra información de la aplicación.
- Contacto: se informa correos electrónicos para realizar diversas consultas.

En el área de registro, los alumnos pueden introducir su libreta universitaria, lo cual les informará su condición con respecto a la cátedra.

En la figura 8.1 de la página 152 se puede observar la interfaz principal de la aplicación.

En la página principal también se encuentran, un menú vertical y un contenedor de imágenes del tipo carrusel, como se puede ver en la figura 8.2 de



Figura 8.1: Página Principal

la página 153, en donde se encuentran los enlaces para seleccionar los applets a ser ejecutados en la parte inferior de la página.

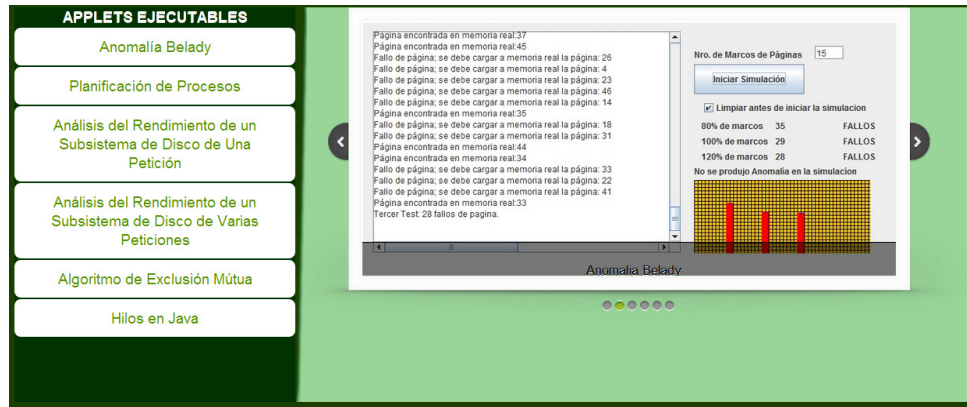


Figura 8.2: Menu Vertical y Carrusel de Imágenes

Capítulo 9

Conclusiones y Líneas Futuras

En primer lugar se hace notar que se han cumplido la totalidad de los objetivos planteados al inicio del presente trabajo.

Con las herramientas utilizadas se consiguió una aplicación amigable al usuario e intuitiva para ser utilizada como parte del trabajo de laboratorio exigido en la materia.

Esta aplicación permite el análisis de los distintos casos de estudio que se explican y desarrollan en el cursado de la cátedra Sistemas Operativos de la FaCENA de la UNNE.

Se ha comprobado la facilidad de uso del sistema con un grupo voluntario de alumnos, quienes han destacado su facilidad de uso al estar basado en un entorno web, accesible desde cualquier navegador y además instalable en equipos propios. De esta manera, los alumnos podrán realizar los laboratorios desde cualquier ordenador con acceso a Internet o descargando el aplicativo a su propio ordenador.

Asimismo se ha verificado que los resultados obtenidos por los distintos componentes del aplicativo son totalmente coherentes con las previsiones teóricas aplicables en los distintos casos.

La ventaja de utilizar herramientas web para el desarrollo de la aplicación permite a la misma ser flexible, con lo cual las actualizaciones resultan más fáciles y rápidas.

Como líneas futuras se plantea, la incorporación de distintos casos de estu-

dio pertenecientes a sistemas operativos convencionales o sistemas operativos distribuidos, como ser, el uso de sistemas expertos para optimizar la gestión de recursos computacionales, distintos casos de sincronización de procesos, sincronización de relojes en sistemas distribuidos, algoritmos de selección de procesos, etc.

La intergración de más casos de estudio a la aplicación significará un beneficio importante para el desarrollo y análisis de los laboratorios de la cátedra de Sistemas Operativos.

Bibliografía

- [1] D. L. La Red Martínez. *Sistemas Operativos*. EUDENE-UNNE, 2004.
- [2] A. S. Tanenbaum. *Sistemas Operativos Modernos - 3E*. Pearson Educación, 1993.
- [3] G. Gagne A. Silberschatz, P. Baer Galvin. *Fundamentos de Sistemas Operativos - 7/E*. McGraw-Hill/Interamericana de España S.A.U., 2006.
- [4] W. Stallings. *Sistemas operativos: aspectos internos y principios de diseño*. Pearson Prentice Hall, 2005.
- [5] H. M. Deitel. *Introducción los Sistema Operativos - 2E*. Addison Wesley Longman, 1993.

Índice alfabético

- administracion de memoria, 25
- algoritmo de planificacion, 4, 8
- algoritmos distribuidos, 126
- almacenamiento primario, 26, 27
- almacenamiento principal, 25
- almacenamiento secundario, 26
- anomalia belady, 27
- anomalia FIFO, 24, 27

- bloqueo, 6
- borradas aleatorias independientes, 51
- busqueda anticipada, 26
- busqueda por demanda, 26

- capacidad cero, 49
- capacidad de colas, 49
- capacidad finita, 49
- capacidad positiva, 49
- carga de trabajo, 77
- categorias de administracion, 25
- clientes por unidad de tiempo, 48
- cola de solicitudes diferidas, 127
- cola FIFO, 78
- colas ilimitadas, 47
- colas limitadas, 47
- colision de clientes, 48
- conjunto de peticiones, 77
- contador, 4
- contextos de ejecucion, 103
- coordinador, 127
- CPU virtual, 4
- creacion de proceso, 6

- cuello de botella, 127

- despachador, 6
- despacho, 6
- destruccion de proceso, 6
- distribucion de los tiempos de servicio, 49
- distribucion de Poisson, 48
- distribucion probabilistica, 47

- estado de proceso, 5
- estados infinitos, 79
- estrategia de busqueda, 26
- estrategia de colocacion, 26
- estrategia de reposicion, 26, 27
- estructura jerarquica de procesos, 6
- exclusion mtua, 127
- exclusion mutua, 126

- fallos de pagina, 27
- fuelle, 47
- fuelle finita, 47
- fuelle infinita, 47

- gestion de memoria, 25

- hilos en java, 103

- interruccion de proceso, 7

- jerarquia de proceso, 5

- lista de bloqueados, 5
- lista de listos, 5

- llegadas, 47
- llegadas de Poisson, 48
- llegadas distribuidas exponencialmente, 48
- memoria principal, 25
- mensajes de control, 127
- modelo de proceso, 4
- multiprocesador, 103
- multiprogramacion, 4
- paginacion FIFO, 24
- paralelismo, 4
- peticion de disco, 79
- planificacion FIFO, 8
- planificacion HRN, 8
- planificacion RNM, 9
- planificacion Round Robin, 8
- poblacion de clientes, 47
- probabilidad de llegadas, 50
- procesador, 3
- proceso, 4
- proceso coordinador, 127
- proceso de Poisson, 50, 51, 78
- proceso de Poisson independiente, 51
- proceso hijo, 6
- proceso padre, 6
- procesos, 5
- procesos concurrentes, 103
- procesos consumidores, 104
- procesos distribuidos, 125
- procesos productores, 104
- procesos secuenciales, 4
- red de colas, 47
- region critica, 126, 127
- registro, 4
- reposicion FIFO, 25, 27
- servidor, 47
- servidor exponencial, 53
- servidores, 47
- servidores en el sistema, 49
- sincronizacion, 126
- sistema de servidores multiples, 50
- sistema distribuido, 125
- sistemas de un solo servidor, 49
- sistemas operativos distribuidos, 125
- sistemas operativos tradicionales, 126
- tabla de proceso, 7
- tasa promedio de llegadas, 48
- tasa promedio de servicio, 78
- teoria de colas, 47
- tiempo de llegadas, 48, 52
- tiempo de servicio, 48
- tiempo de servicio exponencial, 53
- tiempo entre llegadas de primer orden, 50
- tipos de iterrupcion de proceso, 7
- valor esperado, 50, 78
- variable, 4
- variables aleatorias, 47
- varianza, 51