

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

Monografía de Adscripción
Asignatura: Sistemas Operativos

INVESTIGACIÓN DE APPLETS
EN JAVA

APELLIDO Y NOMBRE: *Rodríguez Nelson Fabián*

L.U: 32945

PROF. DIRECTOR: *David Luis La Red Martínez*

LICENCIATURA EN SISTEMAS DE INFORMACIÓN

Corrientes - Argentina

2008

Introducción

Este trabajo práctico de Adscripción consiste en la investigación, análisis y desarrollo de los diferentes casos de estudio propuestos en el SistOper, se han desarrollados 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 su utilización en Internet.

Los casos de estudio desarrollados fueron: Hilos en Java, Algoritmos de Planificación del Procesador y Anomalía Belady, este informe consta de un marco teórico de cada caso de estudio, el código desarrollado y cada una de las ventanas de los applets.

Hilos en JAVA

Concurrencia e Hilos con Java

Los hilos o procesos ligeros son una parte de código o mini programa que puede ser ejecutada 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 es necesario.

Objetivo del Caso de Estudio

El objetivo de este caso de estudio fue el de desarrollar un applet que implementara el problema de “procesos productores y consumidores” y que permitiera generar en un arreglo el seguimiento de dicha simulación y posteriormente efectuar 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.

Descripción de los Algoritmos Utilizados

Respecto del algoritmo de productores y consumidores utilizado como base para el desarrollo del programa del caso de estudio

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 solo 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 solo 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 solo 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.

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 terminar cada ejecución.

El código del applet desarrollado es el siguiente:

```
import java.io.*;
import java.util.*;
import java.util.Date;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.applet.*;
import java.applet.Applet;
import java.math.*;
import java.text.*;
import java.lang.String;
import java.lang.*;
```

```

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=System.currentTimeMillis();

    ArrayList datos;
    public Almacen(ArrayList datos){
        this.datos = datos;
    }

    public synchronized void comprar(int consumidor,
        int cantidad) {
        while (necesitareponer == true) {
            try {
                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.";
            datos.add(c);
            if(stock<=LIMITEREPONER)
                necesitareponer = true;
        }
        notify();
    }

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

                datos.add(c);
            }catch (Exception e) {
                System.out.println("Archivo Hilos: Salida luego de grabar Hilo Productor: " + e);
            }

            necesitareponer = false;
        }

        notify();
    }
}

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);
    }
}

class Productor extends Thread {
    private Almacen almacen;
    private int numero;
    private ArrayList datos;

    public Productor(Almacen a, int numero, ArrayList datos) {
        this.datos = datos;
        almacen = a;
        this.numero = numero;
    }

    public void run() {
        while(!almacen.cerrado){
            if(System.currentTimeMillis()-almacen.hora>almacen.TIEMPOAPERTURA){
                almacen.cerrado=true;

                try {

                    String c;
                    c = "Fin de la ejecución.";

                    datos.add(c);
                }catch (Exception e) {
                    System.out.println("Archivo Hilos: Salida luego de grabar Fin de la ejecución: " + e);
                }
            }
            almacen.reponer(numero);

            try {
                sleep((int)(Math.random() * 100));
            }catch(InterruptedException e) {}
        }
    }
}

class VImaCanvas extends Frame
    implements AdjustmentListener {

    Image imagen1;
    ImaCanvas imaani;
    Panel panel;

    public VImaCanvas(){
        setLayout(new BorderLayout());
        panel = new Panel();
// Agregado de la imagen animada.
        imagen1 = Toolkit.getDefaultToolkit().getImage("imagen1.gif");
        imaani = new ImaCanvas(imagen1);
        panel.add(imaani);
        add("Center",imaani);
    }
}

```

```

        addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            ((Frame)e.getSource()).dispose();
        }
    });

    }
    public void adjustmentValueChanged(AdjustmentEvent evt){
        imaani.repaint();
    }
}

public class AppletHilos extends Applet implements ActionListener{
    private Button btn;
    private VImaCanvas imagen1;

    public void init() {
        btn = new Button("Presione");
        btn.addActionListener(this);
        this.add(btn);
    }

    public void actionPerformed(ActionEvent eve){
        if(eve.getSource() == btn){
            // Creación del menú principal.
            btn.setEnabled(false);
            Hilos menus = new Hilos(this);
            menus.setTitle("Ejecución Concurrente e Hilos con JAVA");
            menus.setVisible(true);
            menus.setBackground(Color.GRAY);
            menus.setSize(new Dimension(850,650));

            // Creación de la imagen dinámica.

            imagen1 = new VImaCanvas();
            imagen1.setTitle("???????");
            imagen1.setVisible(true);
            imagen1.setBackground(Color.orange);
            imagen1.setSize(new Dimension(200,200));
        }
    }

    public void habilitarBoton(){
        btn.setEnabled(true);
        imagen1.dispose();
    }
}

public class AreaTexto extends JFrame {
    public JTextArea areaTexto1, areaTexto2;

    // configurar GUI
    public AreaTexto()
    {
        super( "Demostración de JTextArea" );

        Box cuadro = Box.createHorizontalBox();

        String cadena = "Ésta es una cadena de\ndemostración para\n" +
            "ilustrar cómo copiar texto\nde un área de texto a \n" +
            "otra, utilizando un\nevento externo\n";
    }
}

```

```

// establecer areaTexto1
areaTexto1 = new JTextArea( cadena, 10, 15 );
cuadro.add( new JScrollPane( areaTexto1 ) );

// agregar cuadro al panel de contenido
Container contenedor = getContentPane();
contenedor.add( cuadro ); // colocar en in BorderLayout.CENTER

setSize( 425, 200 );
setVisible( true );

} // fin del constructor de DemoAreaTexto

public static void main( String args[] )
{
    AreaTexto aplicacion = new AreaTexto();
    aplicacion.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}

}

public class Hilos extends Frame
    implements ActionListener {
        VentanaDialogCH ventana;
        SimpleDialogCH dialog;
        TextArea textArea;
        Button boton1;
    private AppletHilos padre;

        // Vector que reemplaza al archivo Hilos.txt
    ArrayList datos;

    public Hilos(AppletHilos padre) {
        // Instanciamos el ArrayList datos
        this.padre = padre;
        this.datos = new ArrayList();
        MenuBar barra;
        Menu cargadatos, ayuda, acerca, salir;
        MenuItem cardatos, ejecuc, listaeje, ayulis, acercade, fin;

        // Menú principal.

        barra = new MenuBar();
        setMenuBar(barra);

        // Agregado de submenú al principal.

        cargadatos = new Menu("Configuración y Ejecución", true);
        barra.add(cargadatos);

        // Creación de opciones del menú.

        cardatos = new MenuItem("Datos de Configuración");
        cardatos.addActionListener(this);
        cargadatos.add(cardatos);
        ejecuc = new MenuItem("Ejecución Concurrente e Hilos");
        ejecuc.addActionListener(this);
        cargadatos.add(ejecuc);
        listaeje = new MenuItem("Lista Resultados de Ejecución");
        listaeje.addActionListener(this);
        cargadatos.add(listaeje);

        // Agregado del submenú al principal.

```

```

        ayuda = new Menu("Ayuda", true);
        barra.add(ayuda);

// Creación de opciones del menú.

        ayulis = new MenuItem("Listar Ayuda");
        ayulis.addActionListener(this);
        ayuda.add(ayulis);

// Agregado del submenú al principal.

        salir = new Menu("Salir", true);
        barra.add(salir);

// Creación de opciones del menú.

        fin = new MenuItem("Salir del Sistema");
        fin.addActionListener(this);
        salir.add(fin);

// Agregado del submenú al principal.

        acerca = new Menu("Acerca de", true);
        barra.add(acerca);

// Creación de opciones del menú.

        acercade = new MenuItem("Acerca del Sistema");
        acercade.addActionListener(this);
        acerca.add(acercade);

// Habilitación del cierre de la ventana.

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                ((Frame)e.getSource()).dispose();
            }
        });
    }

    public void actionPerformed(ActionEvent evt) {
        MenuItem c = (MenuItem) evt.getSource();
        String arg = c.getLabel();
        if(arg.equals("Salir del Sistema")) {
            this.dispose();
        }
        else if(arg.equals("Datos de Configuración")) {

// Para Datos de Configuración.

                // Lee datos de entrada.

                // Crea etiqueta para documento.

                VentanaDialogCH ventana=new VentanaDialogCH(this.datos);
                ventana.setTitle("Carga de Datos de Configuración");
                ventana.setVisible(true);
                ventana.setBackground(Color.gray);
                ventana.setForeground(Color.black);
                ventana.setSize(new Dimension(400,400));
            }
            else if(arg.equals("Ejecución Concurrente e Hilos")) {

// Para Ejecución Concurrente e Hilos.

```

```

        // Lee datos de entrada.

        // Crea etiqueta para documento.

        VentanaDialogECH ventana=new VentanaDialogECH(this.datos);
        ventana.setTitle("Realiza Ejecución Concurrente e Hilos");
        ventana.setVisible(true);
        ventana.setBackground(Color.red);
        ventana.setForeground(Color.yellow);
        ventana.setSize(new Dimension(600,400));
        }
        else if(arg.equals("Lista Resultados de Ejecución")) {

// Para Lista Resultados de Ejecución.

        // Lee datos de entrada.

        // Crea etiqueta para documento.

        VentanaDialogLCH ventana=new VentanaDialogLCH(this.datos);
        ventana.setTitle("Lista Resultados de Ejecución Concurrente e Hilos");
        ventana.setVisible(true);
        ventana.setBackground(Color.green);
        ventana.setForeground(Color.orange);
        ventana.setSize(new Dimension(600,400));
        }
        else if(arg.equals("Listar Ayuda")) {

// Para Listar Ayuda.

        // Lee datos de entrada.

        // Crea etiqueta para Listar Ayuda.

        VentanaDialogA ventana=new VentanaDialogA();
        ventana.setTitle("Lista Ayuda");
        ventana.setVisible(true);
        ventana.setBackground(Color.cyan);
        ventana.setForeground(Color.magenta);
        ventana.setSize(new Dimension(600,400));
        }
        else if(arg.equals("Acerca del Sistema")) {

// Para Acerca del Sistema.

        // Lee datos de entrada.

        // Crea etiqueta para Acerca del Sistema.

        VentanaDialogAS ventana=new VentanaDialogAS();
        ventana.setTitle("Acerca del Sistema");
        ventana.setVisible(true);
        ventana.setBackground(Color.orange);
        ventana.setForeground(Color.blue);
        ventana.setSize(new Dimension(600,400));
        }

    }

    public void dispose(){
        padre.habilitarBoton();
        super.dispose();
    }
}

class ImaCanvas extends Canvas{

```

```

Image img;
int x,y;

public ImaCanvas(Image imagen1){
    img=imagen1;
    x=0;
    y=0;
}

public void paint(Graphics g){
    g.drawImage(img,0,0,this);
}
}

class SimpleDialogCH extends Dialog implements ActionListener{
    TextField tstock, tmaxcompra, tlimitereponer, tcantidadreponer, ttiempoapertura;
    VentanaDialogCH parent;
    Button b1;
    public String stock;
    public String MAXCOMPRA;
    public String LIMITEREPONER;
    public String CANTIDADREPONER;
    public String TIEMPOAPERTURA;

    public SimpleDialogCH(Frame fr,String titulo){
        super(fr,titulo,true);
        parent=(VentanaDialogCH)fr;

        Panel p1=new Panel();
        Label etiqueta1=new Label("Número inicial de buffers ocupados (0):");
        p1.add(etiqueta1);
        tstock=new TextField("0",4);
        p1.add(tstock);
        add("Center",p1);
        Label etiqueta2=new Label("Número máximo de buffers leídos por vez (30):");
        p1.add(etiqueta2);
        tmaxcompra=new TextField("30",4);
        p1.add(tmaxcompra);
        add("Center",p1);
        Label etiqueta3=new Label("Número mínimo de buffers ocupados antes de grabar más (40):");
        p1.add(etiqueta3);
        tlimitereponer=new TextField("40",4);
        p1.add(tlimitereponer);
        add("Center",p1);
        Label etiqueta4=new Label("Número de buffers grabados por vez (50):");
        p1.add(etiqueta4);
        tcantidadreponer=new TextField("50",4);
        p1.add(tcantidadreponer);
        add("Center",p1);
        Label etiqueta5=new Label("Número de milisegundos de la ejecución (2000):");
        p1.add(etiqueta5);
        ttiempoapertura=new TextField("2000",4);
        p1.add(ttiempoapertura);
        add("Center",p1);

        Panel p2=new Panel();
        p2.setLayout(new FlowLayout(FlowLayout.RIGHT));
        b1=new Button("Cancelar");
        b1.addActionListener(this);
        Button b2=new Button("Aceptar");
        b2.addActionListener(this);
        p2.add(b1);
        p2.add(b2);
        add("South",p2);
        setSize(500,200);
    }
}

```

```

public void actionPerformed(ActionEvent evt){
    String str=evt.getActionCommand();
    if(str.equals("Aceptar"))
        parent.setText1(tstock.getText());
    stock = tstock.getText();
    parent.setText2(tmaxcompra.getText());
    MAXCOMPRA = tmaxcompra.getText();
    parent.setText3(tlimitereponer.getText());
    LIMITEREPONER = tlimitereponer.getText();
    parent.setText4(tcantidadreponer.getText());
    CANTIDADREPONER = tcantidadreponer.getText();
    parent.setText5(ttiempoapertura.getText());
    TIEMPOAPERTURA = ttiempoapertura.getText();
    setVisible(false);
}
}

class VentanaDialogA extends Frame implements ActionListener{
    TextArea textArea;

    public VentanaDialogA(){
        textArea=new TextArea(5,40);
        textArea.setEditable(false);
        add("Center",textArea);

        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                setVisible(false);
            }
        });

    try {

        String          asterisc          =          "\n
        *****";

        textArea.setFont(new Font("Times", 1, 11));
        textArea.setForeground(Color.blue);

        textArea.append("\n ");
        textArea.append(asterisc);
        textArea.append("\n ");
        textArea.append("\n Ayuda para la ejecución concurrente e hilos con JAVA.");
        textArea.append("\n
        *****");

        textArea.append("\n ");
        textArea.append("\n El ejemplo contempla el problema de productores / consumidores:");
        textArea.append("\n * Los hilos productores graban (y ocupan) buffers en un pool");
        textArea.append("\n de buffers disponibles.");
        textArea.append("\n * Los hilos consumidores leen (y liberan) buffers del pool");
        textArea.append("\n de buffers disponibles.");
        textArea.append("\n ");
        textArea.append("\n Las opciones previstas contemplan lo siguiente:");
        textArea.append("\n * Carga de datos de configuración de la ejecución.");
        textArea.append("\n * Ejecución propiamente dicha.");
        textArea.append("\n * Listado de los resultados por pantalla.");
        textArea.append("\n ");
        textArea.append("\n Los datos se cargan interactivamente por pantalla, ");
        textArea.append("\n donde se sugieren valores de referencia.");
        textArea.append("\n ");
        textArea.append("\n Los resultados se guardan en un arreglo alojado en memoria,");
        textArea.append("\n los mismos se pierden al cerrar el applet.");
        textArea.append("\n ");

        textArea.append("\n ");
        textArea.append(asterisc);
        textArea.append("\n ");
    }
}

```

```

catch (java.lang.Exception ex) {

    // Atrapa otro tipo de excepciones y las despliega.

    ex.printStackTrace ();

}

}

public void actionPerformed(ActionEvent evt){
    String str=evt.getActionCommand();
}

}

class VentanaDialogAS extends Frame implements ActionListener{
    TextArea textArea;

    public VentanaDialogAS(){
        textArea=new TextArea(5,40);
        textArea.setEditable(false);
        add("Center",textArea);

        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                setVisible(false);
            }
        });

    try {

        String          asterisc          =          "\n
        *****
        **";

        textArea.setFont(new Font("Times", 1, 11));
        textArea.setForeground(Color.black);

        textArea.append("\n ");
        textArea.append(asterisc);
        textArea.append("\n ");

        textArea.append("\n Acerca del sistema de concurrencia e hilos con JAVA.");
        textArea.append("\n
        *****
        **");

        textArea.append("\n ");
        textArea.append("\n Trabajo práctico de Adscripción de la Catedra Sistemas Operativos.");
        textArea.append("\n Licenciatura en Sistemas de Información.");
        textArea.append("\n Universidad Nacional del Nordeste (Argentina - 2007).");
        textArea.append("\n ");

        textArea.append("\n ");
        textArea.append(asterisc);
        textArea.append("\n ");

    }

    catch (java.lang.Exception ex) {

        // Atrapa otro tipo de excepciones y las despliega.

        ex.printStackTrace ();

    }

}

```

```

        public void actionPerformed(ActionEvent evt){
            String str=evt.getActionCommand();
        }
    }

class VentanaDialogCH extends Frame implements ActionListener{
    ArrayList datos;
    SimpleDialogCH dialog;
    TextArea textArea;
    Button boton1;

    // Constructor modificado para funcionar en Applet
    // el cual recibe un ArrayList para el intercambio de datos.
    public VentanaDialogCH(ArrayList datos){
        this.datos = datos;
        textArea=new TextArea(5,40);
        textArea.setEditable(false);
        add("Center",textArea);
        boton1=new Button("Diálogo p/ configuración");
        boton1.addActionListener(this);
        Panel panel=new Panel();
        panel.add(boton1);
        add("South",panel);
        panel.setBackground(Color.yellow);

        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                setVisible(false);
            }
        });
    }

    public void actionPerformed(ActionEvent evt){
        String str=evt.getActionCommand();
        if(str.equals("Diálogo p/ configuración")){
            if(dialog==null)
                dialog=new SimpleDialogCH(this,"Configuración");
            //dialog.show(); este metodo esta deprecated
            dialog.setVisible(true);
        }
    }

    public void setText1(String stock){
        try {
            textArea.setFont(new Font("Times", 1, 11));
            textArea.setForeground(Color.orange);

            String asterisc = "\n*****";
            textArea.append(asterisc);
            textArea.append("\nNúmero inicial de buffers ocupados: " + stock + ".");
        } catch (java.lang.Exception ex) {
            // Atrapa excepciones y las despliega.
            ex.printStackTrace ();
        }
    }

    //Este bloque reemplaza el uso del archivo para el intercambio de datos
    datos.clear();
    datos.add(stock);
}

    public void setText2(String maxcompra){
        try {
            textArea.append("\nNúmero máximo de buffers leídos por vez: " + maxcompra + ".");
        } catch (java.lang.Exception ex) {
    }

```

```

        // Atrapa excepciones y las despliega.
        ex.printStackTrace ();
    }

    datos.add(maxcompra);
}

    public void setText3(String limitereponer){
try{
    textArea.append("\nNúmero mínimo de buffers ocupados antes de grabar más: " + limitereponer +
".");
        }catch (java.lang.Exception ex) {
// Atrapa excepciones y las despliega.
ex.printStackTrace ();
        }

    datos.add(limitereponer);
}

    public void setText4(String cantidadreponer){
try {
    textArea.append("\nNúmero de buffers grabados por vez: " + cantidadreponer + ".");
        }catch (java.lang.Exception ex) {
// Atrapa excepciones y las despliega.
ex.printStackTrace ();
        }

    datos.add(cantidadreponer);
}

    public void setText5(String tiempoapertura){
try {
String asterisc = "\n*****";
    textArea.append("\nNúmero de milisegundos de la simulación: " + tiempoapertura + ".");
    textArea.append(asterisc);
    textArea.append("\n ");
        }catch (java.lang.Exception ex) {
// Atrapa excepciones y las despliega.
ex.printStackTrace ();
        }

    System.out.println("Archivo Hilos: Salida en grabación de parámetros: " + e);
    }*/
    datos.add(tiempoapertura);
}
}

class VentanaDialogECH extends Frame implements ActionListener{
    TextArea textArea;
    ArrayList datos;

//Constructor original
// Modificado para compartir los datos necesarios
// a travez del ArrayList datos
public VentanaDialogECH(ArrayList datos){
    this.datos = datos;
        textArea=new TextArea(5,40);
        textArea.setEditable(false);
        add("Center",textArea);

    String stock, maxcompra, limitereponer, cantidadreponer, tiempoapertura;
    stock = ""; maxcompra = ""; limitereponer = "";
    cantidadreponer = ""; tiempoapertura = "";
}

```

```

        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                setVisible(false);
            }
        });

stock = (String)datos.get(0);
maxcompra = (String)datos.get(1);
limitereponer = (String)datos.get(2);
cantidadreponer = (String)datos.get(3);
tiempoapertura = (String)datos.get(4);
try {
    Almacen a = new Almacen(this.datos);
    Productor p1 = new Productor(a,1,this.datos);
    Productor p2 = new Productor(a,2,this.datos);
    Consumidor c1 = new Consumidor(a,1);
    Consumidor c2 = new Consumidor(a,2);
    Consumidor c3 = new Consumidor(a,3);

    a.stock = Integer.parseInt(stock);
    a.MAXCOMPRA = Integer.parseInt(maxcompra);
    a.LIMITEREPONER = Integer.parseInt(limitereponer);
    a.CANTIDADREPONER = Integer.parseInt(cantidadreponer);
    a.TIEMPOAPERTURA = Integer.parseInt(tiempoapertura);

    p1.start();
    p2.start();
    c1.start();
    c2.start();
    c3.start();
} catch (java.lang.Exception ex) {
    // Atrapa excepciones y las despliega.
    ex.printStackTrace ();
}

try {

    String c, d, asterisc;

    asterisc = "\n*****";
    d = "\n ";

    textArea.setFont(new Font("Times", 1, 11));
    textArea.setForeground(Color.red);

    textArea.append(asterisc);
    textArea.append("\nInicio de la ejecución concurrente con hilos.");
    textArea.append("\nLos datos de configuración son los siguientes: ");
    c = (String)datos.get(0);
    textArea.append("\nNúmero inicial de buffers ocupados: " + c + ".");
    c = (String)datos.get(1);
    textArea.append("\nNúmero máximo de buffers leídos por vez: " + c + ".");
    c = (String)datos.get(2);
    textArea.append("\nNúmero mínimo de buffers ocupados antes de grabar más: " + c + ".");
    c = (String)datos.get(3);
    textArea.append("\nNúmero de buffers grabados por vez: " + c + ".");
    c = (String)datos.get(4);
    textArea.append("\nNúmero de milisegundos de la ejecución: " + c + ".");
    textArea.append(asterisc);
    textArea.append(d);

} catch (Exception e) {
    System.out.println("Archivo Hilos: Salida luego de leer parámetros para desplegarlos: " + e);
}
}

```

```

        public void actionPerformed(ActionEvent evt){
            String str=evt.getActionCommand();
        }
    }

class VentanaDialogLCH extends Frame implements ActionListener{
    TextArea textArea;
    ArrayList datos;

    //Constructor original
    // Modificado para compartir los datos necesarios
    // a travez del ArrayList datos
    public VentanaDialogLCH(ArrayList datos){
        this.datos = datos;
        textArea=new TextArea(5,40);
        textArea.setEditable(false);
        add("Center",textArea);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                setVisible(false);
            }
        });

        try {

            String c, d, asterisc;

            asterisc = "\n*****";
            d = "\n ";

            textArea.setFont(new Font("Times", 1, 11));
            textArea.setForeground(Color.red);

            textArea.append(asterisc);
            textArea.append("\nInicio de la ejecución concurrente con hilos.");
            textArea.append("\nLos datos de configuración son los siguientes: ");
            c = (String)datos.get(0);
            textArea.append("\nNúmero inicial de buffers ocupados: " + c + ".");
            c = (String)datos.get(1);
            textArea.append("\nNúmero máximo de buffers leídos por vez: " + c + ".");
            c = (String)datos.get(2);
            textArea.append("\nNúmero mínimo de buffers ocupados antes de grabar más: " + c + ".");
            c = (String)datos.get(3);
            textArea.append("\nNúmero de buffers grabados por vez: " + c + ".");
            c = (String)datos.get(4);
            textArea.append("\nNúmero de milisegundos de la ejecución: " + c + ".");
            textArea.append(asterisc);
            textArea.append(d);

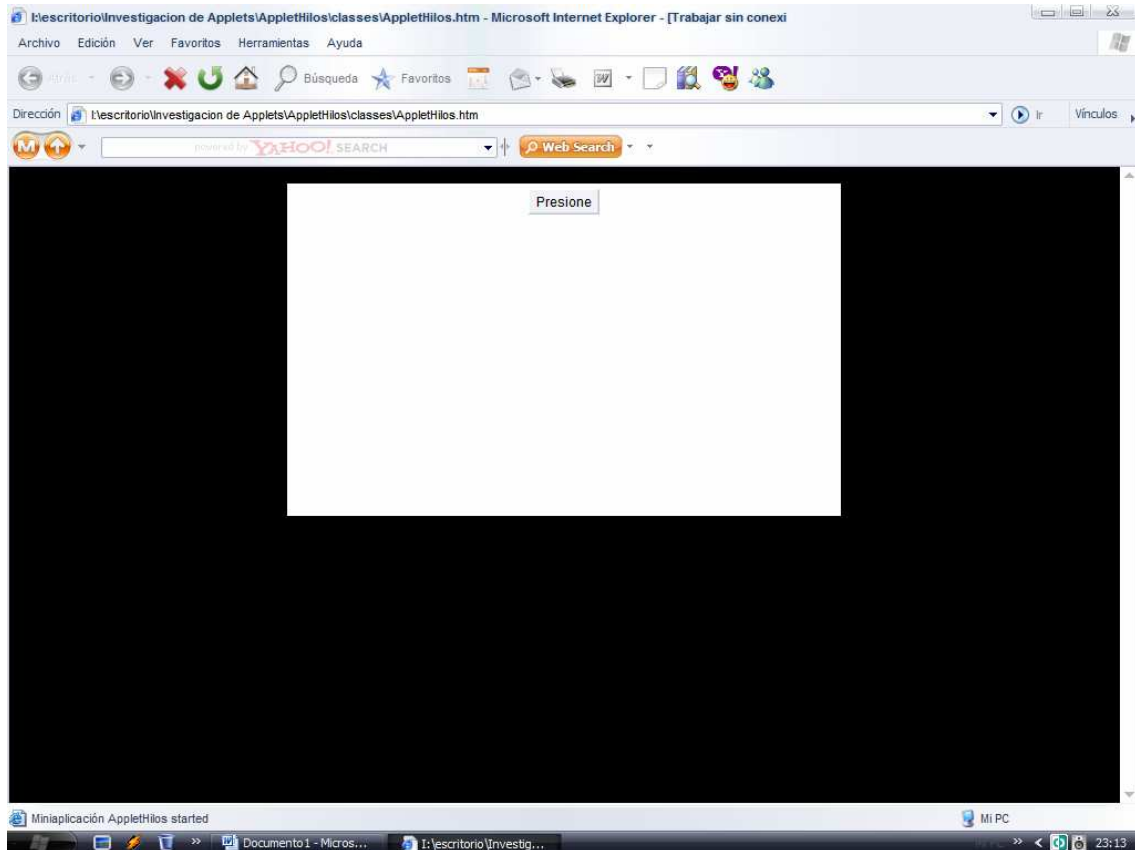
            Iterator i = datos.iterator();
            while(i.hasNext()){
                c = (String)i.next();
                textArea.append(c);
            }
            textArea.append(d);
        } catch (Exception e) {
            System.out.println("Archivo Hilos: Salida luego de leer parámetros para desplegarlos: " + e);
        }
    }

    public void actionPerformed(ActionEvent evt){
        String str=evt.getActionCommand();
    }
}

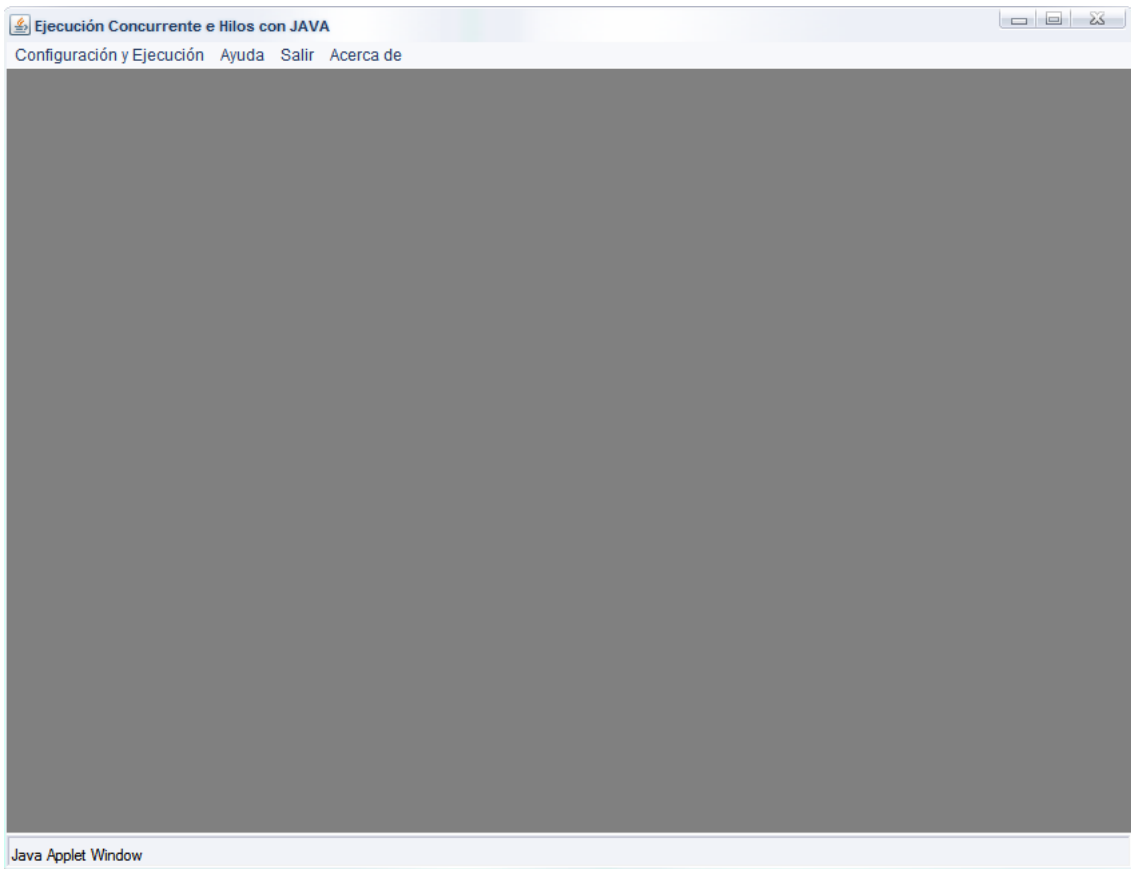
```

Datos y Ejecuciones

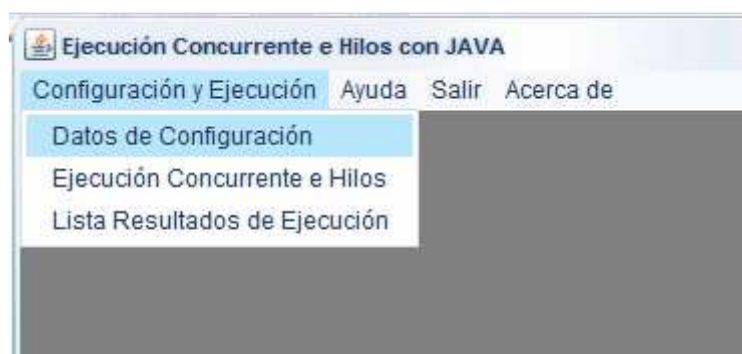
Los resultados detallados de las ejecuciones se muestran paso a paso en pantalla, como se muestra a continuación:



Pantalla Principal para acceder al Applet



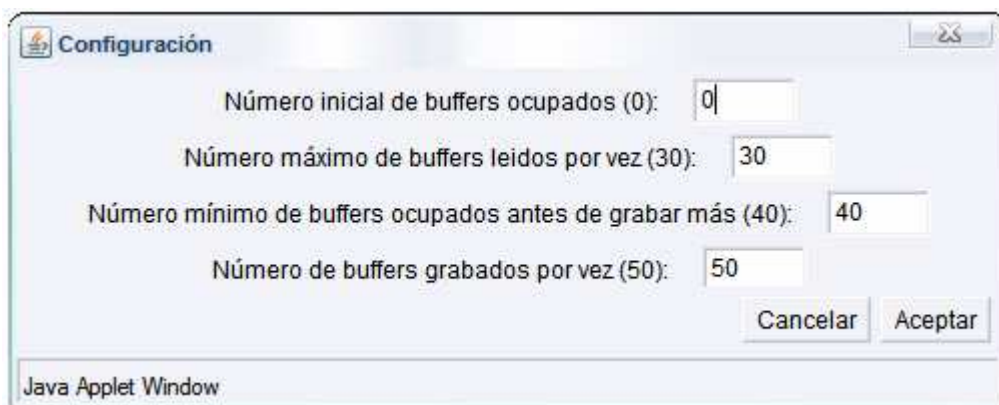
Pantalla Principal del Applet



Selección del Menú para la configuración de los datos



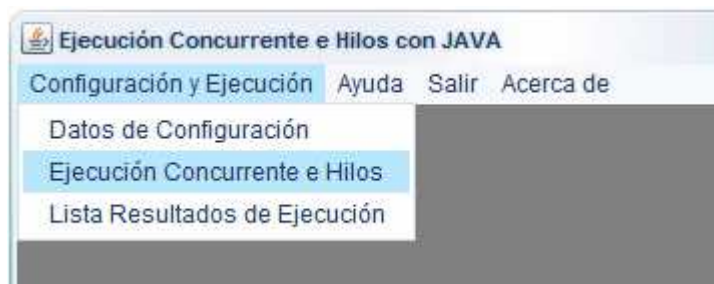
Pantalla para cargar los datos de la configuración que deseamos.



Pantalla con los datos por defecto.



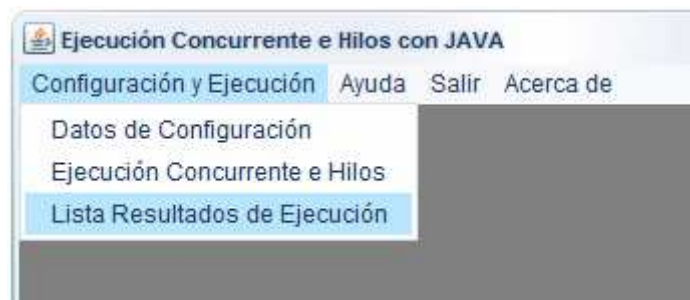
Datos con los cuales se va a realizar la simulación.



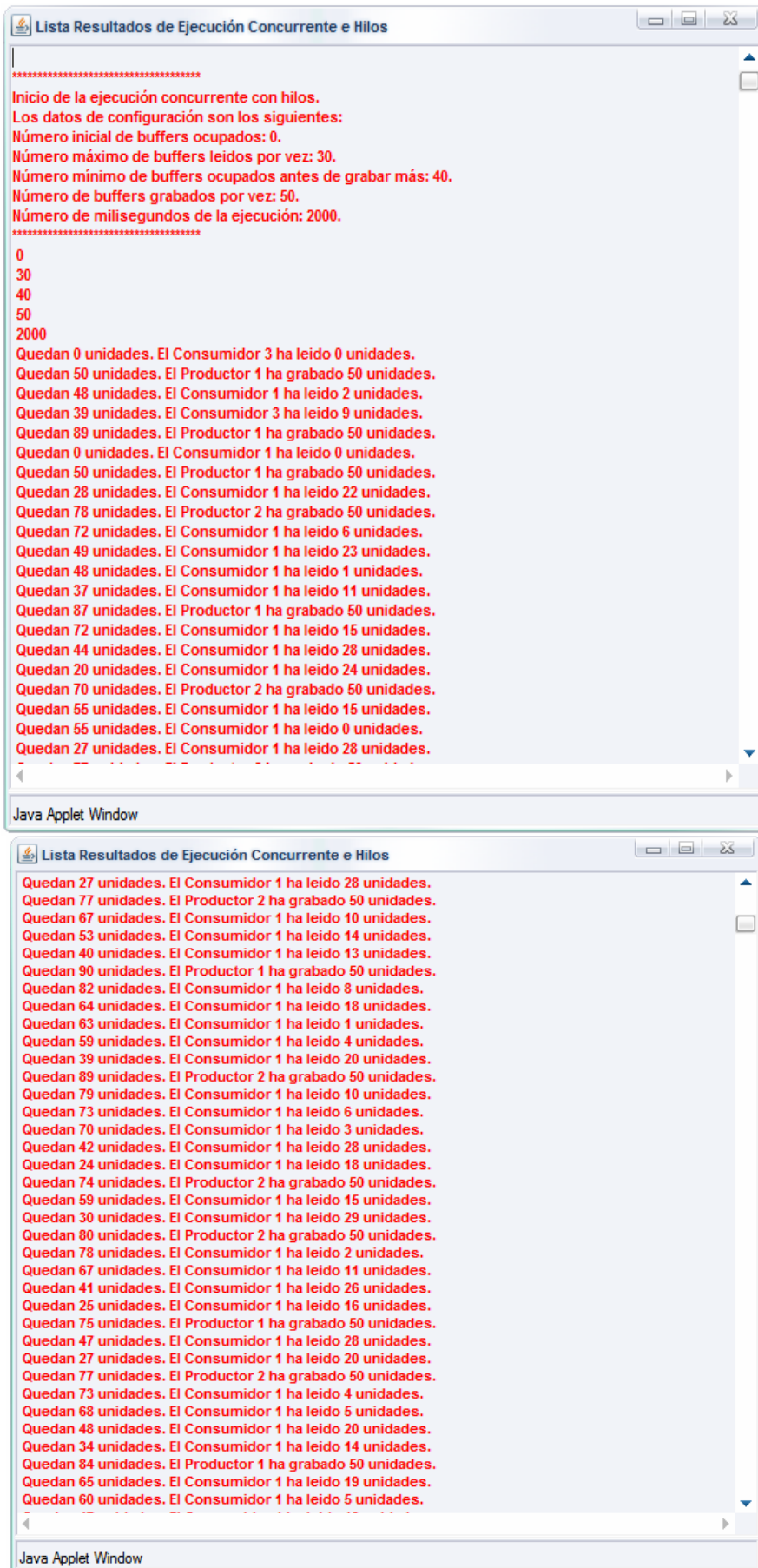
Selección del Menú para realizar la ejecución del programa.



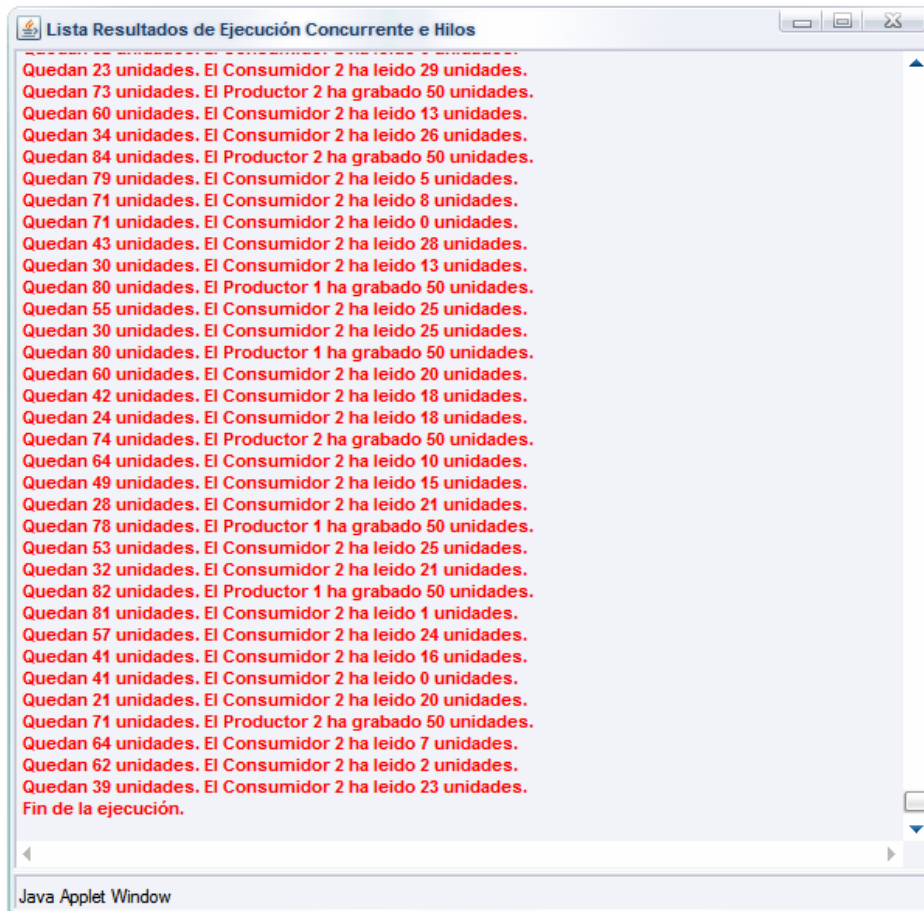
Se realiza la ejecución concurrente con los datos mostrados en pantalla.



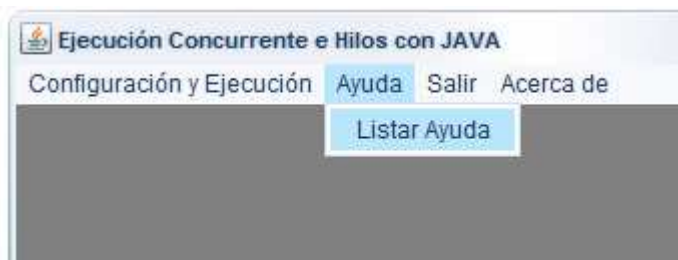
Selección del Menú para ver los resultados de la simulación.



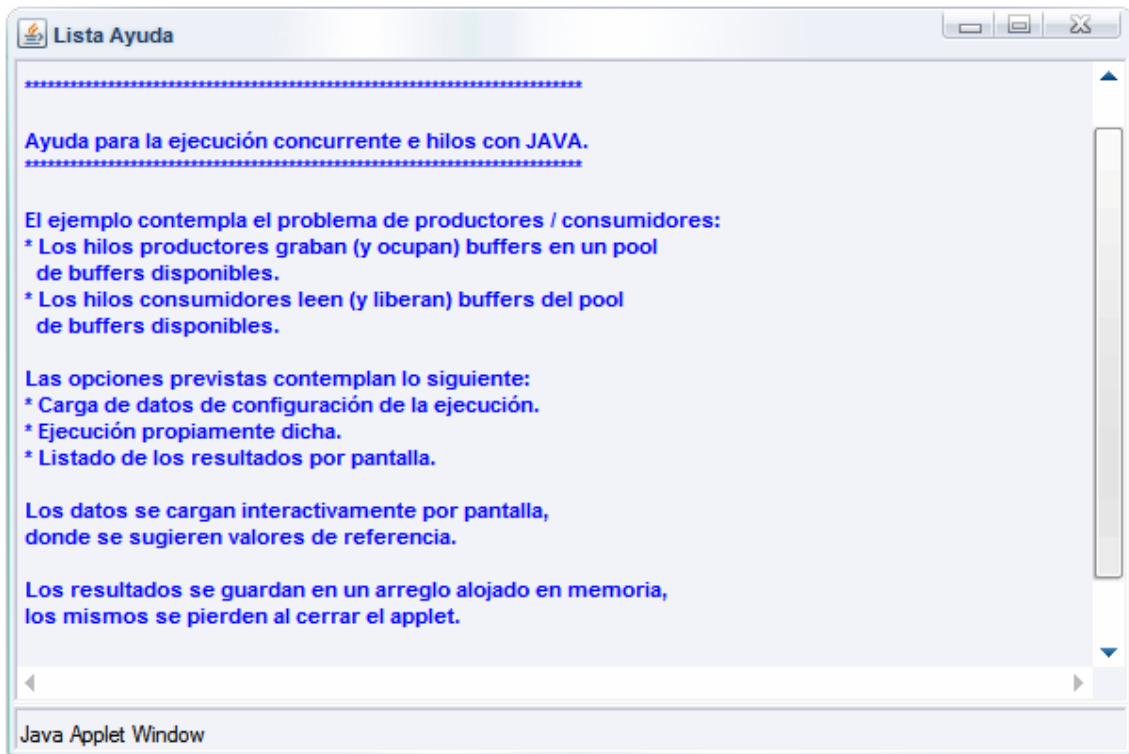
Pantalla con los resultados de la ejecución.



Pantalla con los resultados de la ejecución.



Selección del Menú de Ayuda.



Pantalla de Ayuda para la ejecución del applet.

Conclusión

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 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, es decir el caso de procesos productores y procesos consumidores solo un caso genérico de la problemática antes mencionada.

Algoritmos de Planificación del Procesador

Introducción

Una de las muchas y muy variadas posibles aplicaciones de la P.O.O. (programación orientada a objetos), está en el desarrollo de algoritmos que implementen estrategias de administración de recursos por parte del Sistema Operativo.

Como parte de las estrategias antes mencionadas, podemos considerar las de administración o asignación del procesador, es decir aquéllas 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.

Objetivo del Caso de Estudio

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

A diferencia del la simulación que se puede observar en el SistOper el applet desarrollado no realiza la grabación de la simulación en un archivo, sino que muestra los datos del resultado de la simulación en una pantalla dentro del mismo applet en la que se incorpora para cada simulación efectuada, un resumen de los resultados obtenidos referidas a cada una de las estrategias consideradas, lo que permite su análisis y discusión posterior. Como los resultados solo son mostrados en la ventana del applet, estos se pierden al cerrar el applet.

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**.

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.

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 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 que puede tener un proceso son 3:

- **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.

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”.

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”.

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” 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”.

La “destrucción” 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 recepción 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.

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.
- **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.
- **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.
- **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 (quantum) 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.

El código 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.setToolTipText("Ingrese la cantidad de procesos a utilizar en la simulaci\u2666n.");

        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("Acerca 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 Mster. David Luis La Red Martínez,\nen la Facultad de Ciencias Exactas y Naturales y Agrimensura,\ndependiente de la Universidad Nacional del Nordeste (UNNE),\npor los alumnos Nelson Fabián Rodríguez (Adscripto a la Catedra)\ny Anibal Sebastian Rolando (Colaborador)");

```

jTextArea1.setEditable(false);
ven.setTitle("Acerca 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()
            .addGap(475,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jScrollPane1,
                        javax.swing.GroupLayout.PREFERRED_SIZE,
                        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.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                        .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)
                                    .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)))
            .addGap(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;
    }
}

```

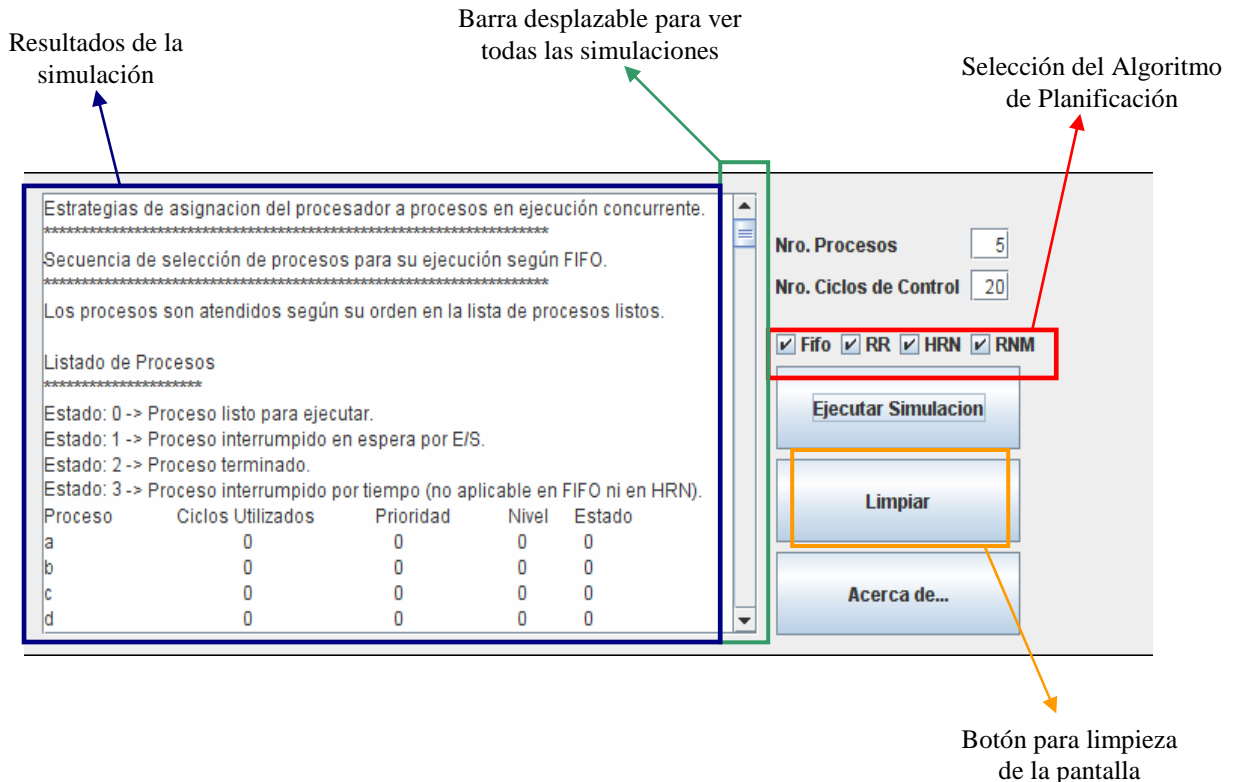
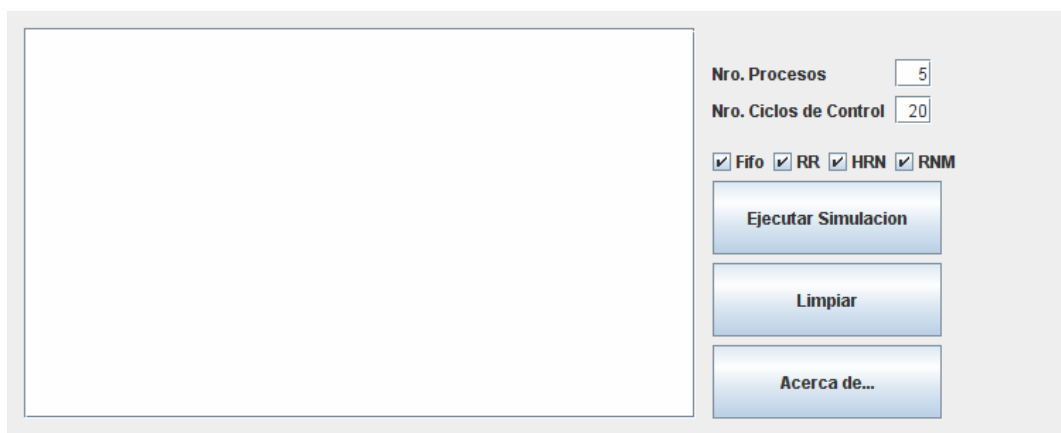
```
    }  
  
    if(i > 3){  
        throw new RuntimeException("Esta mal el random...");  
    }  
  
    this.estado = i;  
}  
  
public void reInicio(){  
    this.estado = 0;  
    this.cuenta = 0;  
    this.pri = 0;  
    this.nivel = 0;  
}  
}
```

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, estos son el número de ciclos de control (20) y el número de procesos (5) que intervendrán en la simulación.

Los resultados detallados de las ejecuciones se muestran paso a paso en la pantalla del applet con un resumen al final de cada una de las estrategias de planificación.



Anomalía BELADY

Introducción

Como los casos de estudio anteriores, este caso fue desarrollado en un applet para que pueda ser utilizado en Internet, a diferencia de los casos anteriores en donde los algoritmos eran similares a los desarrollados en el SistOper, el desarrollo del algoritmo aplicado a la simulación de requerimientos de paginación de memoria virtual fue creado 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

Objetivo

Conforme a lo antes indicado, el objetivo del caso de estudio desarrollado fue el de codificar un programa en un applet en JAVA que efectuara el testeo 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.

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.

- ❖ “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.
- ❖ “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.
- ❖ “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.

Reposición de Página por el Sistema de Primero en Entrar – Primero en Salir (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 FIFO”:

- ❖ 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.

Programa Desarrollado

El applet desarrollado posee las siguientes características:

Recibe como entrada por pantalla los datos de configuración de la simulación referidos a:

- ❖ 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.
- ❖ Conclusión respecto de 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("El 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 pagina requerida en los marcos

            if(this.marcos[j] == this.requerimientos[i]){

```

```

        encontro = true;
        j = this.marcos.length;
    }
}

if(!encontro){// Si la pagina 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 pagina." ) ;
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 pagina 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(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.ComponentPlacement.RELATED)
            .addComponent(jLabel6)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel7)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jLabel12, javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .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))
    );
} // </editor-fold>
private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    p.setMarcos(Integer.parseInt( jTextField1.getText()));
    p.setRequerimientos((int)(Integer.parseInt(jTextField1.getText()) * 3));
    p.simular(this.jCheckBox1.isSelected());
}
// Variables declaration - do not modify
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;
private javax.swing.JLabel jLabel9;
private javax.swing.JPanel jPanel1;

```

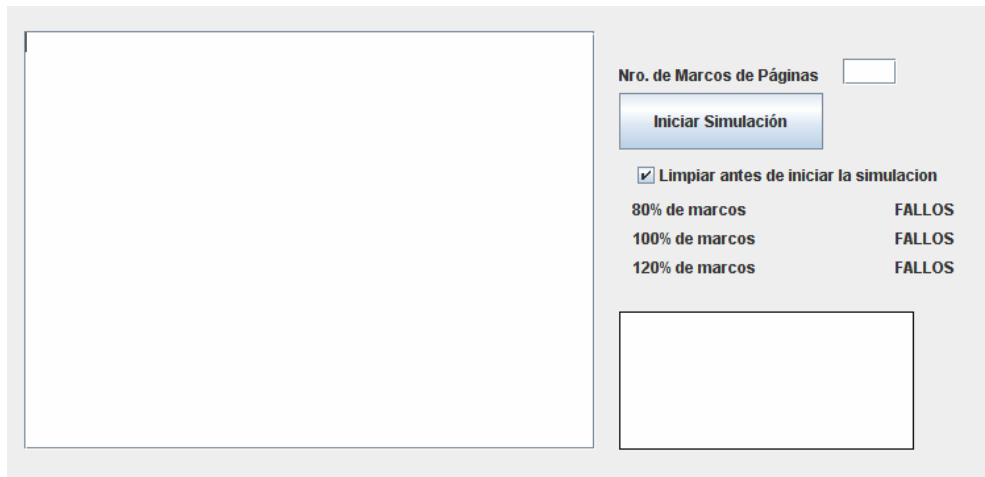
```

private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JTextArea jTextArea1;
private javax.swing.JTextField jTextField1;
// End of variables declaration
}

```

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.



Pantalla Principal del Applet

Informe detallado de la simulación

Cantidad de marcos de página ingresados

Resumen de la simulación

80% de marcos	36	FALLOS
100% de marcos	33	FALLOS
120% de marcos	29	FALLOS

No se produjo Anomalia en la simulación

Grafico de los resultados de la simulación

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 Anomalía Belady o Anomalía FIFO es una situación especial que no se observa en ninguna de las simulaciones que se realizan, esto ratifica lo que se puede pensar intuitivamente, que al aumentar la cantidad de celdas o marcos de pagina obtendremos menos fallos de pagina en un esquema de paginación FIFO.

También se ha observado que al aumentar considerablemente las celdas o marcos de pagina 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.