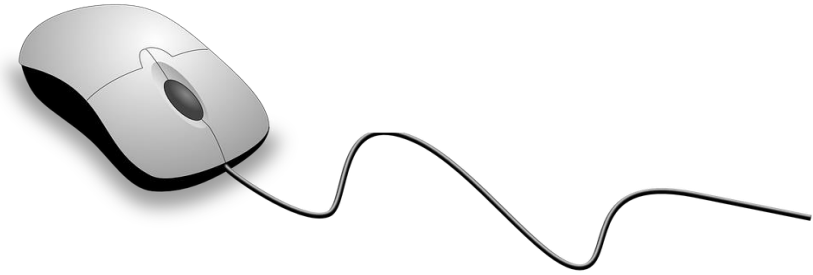
Proyecto Sistemas Operativos I
Simulador de Gestor de Procesos
Manual técnico

PERIODO 1
2020

INTEGRANTES:

- David Alexander Cardenas Almendares 20121003387
- Cristian Alexander Martínez Ochoa 20131015700
- Ariel Isaí Turcios García 20131006640
- Franklin Romario Chavarría Laínez 20131007637
- Emerson Joel Amador 20121016684



Manual Técnico

Índice

1. Objetivos.

Se ha creado dicho documento con el propósito de mostrar cómo fue diseñada la aplicación, Y al mismo tiempo dar referencias de como interactuar con el programa.

Todo esto para futuras actualizaciones o para darle mantenimiento por otro programador. Se especifica su creación, proceso de instalación, código fuente etc…

1.1 Objetivos específicos.

1. Comprende los pasos a seguir para gestionar los procesos por parte del sistema operativo.

2. Desarrollar un gestor de procesos en base al modelo de los 5 estados.

2. Alcance.

Este documento va dirigido a un programador con conocimientos medios de lenguaje java además de los conocimientos de la clase de sistemas operativos uno relacionado con los con el modelo de los 5 estados.

3. Requerimientos técnicos.

Software:

- Java, JDK
- IDE NetBeans o Eclipse
- Computadora con Windows 7 o superior

Hardware:

- Una computadora completa (bocinas no necesarias)

3.1 Requerimientos mínimos de hardware.

- Procesador: Intel Inside 1.5 Ghz
- Memoria RAM (mínimo): 512 mb
- Disco duro: 10gb

3.2 Requerimientos mínimos de software

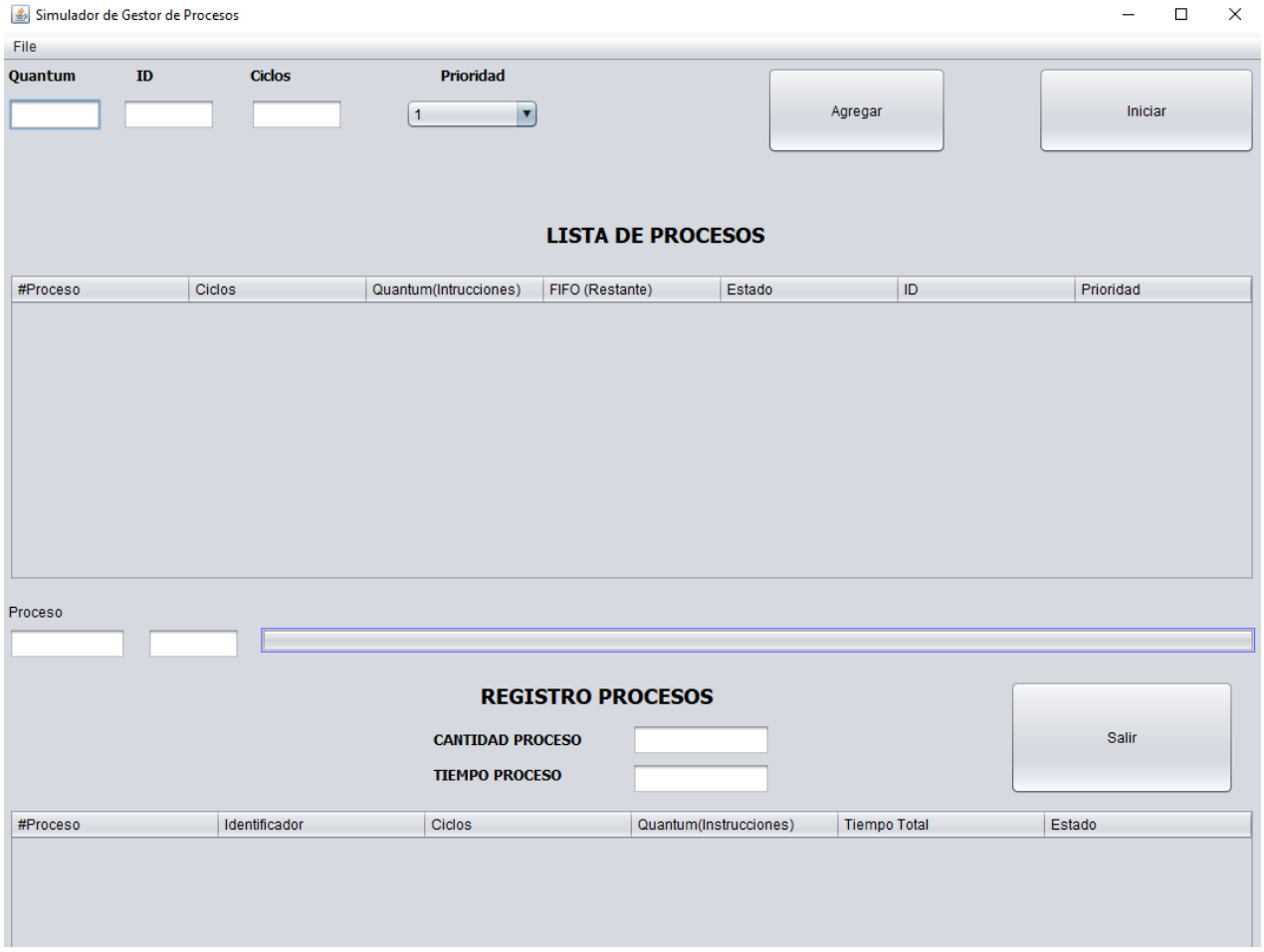
- Privilegios de administrador: si
- Sistema operativo: Windows 7 o superior

4. Herramientas utilizadas para el desarrollo.

- IDE NetBeans 8.2
- Documentación de java en línea
- Editor de texto
- Documentación de la clase acerca de los 5 estados

5. Instalación.

No requiere de una instalación ya que se es un archivo ejecutable



Pantalla principal de la aplicación

6. Configuración.

No hay una configuración general, la aplicación en si viene configurada y los comandos son intuitivos para el usuario.

7. Diccionario de clases y métodos.

Se utilizó una única clase llamada **Procesar.java** (es un Jframe, para manejo de interfaces graficas en java) esta contiene todas las variables, así como también los métodos utilizados en el desarrollo del proyecto.

```
import java.awt.Color;
import javax.swing.JOptionPane;
import static java.lang.Integer.parseInt;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.table.DefaultTableModel;
import Atxy2k.CustomTextField.RestrictedTextField;
```

** Se deben de importar todas estas bibliotecas para un correcto funcionamiento de la aplicación.

```
public class Procesar extends javax.swing.JFrame {

    int Contador;//Contador del total de procesos que se van ingresando
    int NProceso;//Carga el número de procesos ejecutándose
    int IdProceso;//Carga el ID del proceso
    int Rafaga=0;//Carga la ráfaga en ejecución
    int Quantum=0;//Carga el Intervalo - Periodo (quantum) en ejecución
    int ResiduoRafaga=0;//Carga el residuo en ejecución
    int TiempoProceso=0;//Carga el tiempo que se dura procesando
    int ValorBarra;//Carga el progreso de la Barra
    int CantidadProcesos;//Número de procesos terminados

.
.
.
}
```

```
/**
    * Creates new form Procesar
    */
   public Procesar() {
      initComponents();
      jTIngreso.setBackground(Color.CYAN);
      jTIngreso.setForeground(Color.blue);
      jTFinal.setBackground(Color.green);
      //jTFinal.setBackground(Color.red);
      jTFCapturaQuantum.grabFocus();
      RestrictedTextField r = new RestrictedTextField(jTFCapturaID);
      r.setOnlyNums(true);
      r.setLimit(4);
      RestrictedTextField r2 = new RestrictedTextField(jTFCapturaRafaga);
      r2.setOnlyNums(true);
      r2.setLimit(3);
   }
```

** Este método se utiliza para colorear a los procesos, además determina los límites de cada uno de los jTF, por último, se instancian los TextField.

Método ingresar(): se crea un modelo con referencia al modelo por defecto de la tabla jTIngreso, se encuentra una variable contadora la cual lleva el conteo de la cantidad de procesos a ingresar la cual aumenta en cantidades de 1 por cada proceso a ingresar, crea un objeto con la cantidad de columnas igual a la cantidad de campos del modelo de la tabla en las cuales se ingresa la cantidad que el usuario desea en los textFile incluyendo el valor que toma la variable contador en ese instante y un String sobre el estado de dicho proceso, luego se agrega a la tabla modelo el objeto y luego se agrega a la tabla jTIngreso la tabla modelo, la última parte de código representa la forma en que borra el dato ingresado en el textFile para ser utilizado nuevamente con un dato nuevo.

```
public void Ingresar(){ //Ingresar proceso a la tabla
   DefaultTableModel modelo=(DefaultTableModel) jTIngreso.getModel();
   Contador ++;
   Object[] miTabla = new Object[7];
   miTabla[0]= Contador;
   miTabla[1]= jTFCapturaRafaga.getText();
   miTabla[2]= jTFCapturaQuantum.getText();
   miTabla[3]= jTFCapturaRafaga.getText();
   miTabla[4]= "Listo";
   miTabla[5]= jTFCapturaID.getText();
   miTabla[6]= jTFCapturaPrioridad.getSelectedItem();
   modelo.addRow(miTabla);
   jTIngreso.setModel(modelo);
   jTFCapturaID.setText(null);
   jTFCapturaID.grabFocus();
   jTFCapturaRafaga.setText(null);

}
```

Botón Agregar:


```
    private void jBAgregarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        if((Integer.parseInt(jTFCapturaRafaga.getText()))<=100){
            Ingresar();
            jTFCapturaQuantum.setEditable(false);
        }else{
            JOptionPane.showMessageDialog(null, "Las Rafagas de ciclo no pueden ser mayores
  de 100");
            jTFCapturaRafaga.setText(null);
            jTFCapturaRafaga.grabFocus();

        }

    }
```

Método jBAgregarActionPerformed (botón): es una condicional si la ráfaga es menor a 100 llama al método ingresar y luego bloquea el TextField jTCapturaQuantum ya que utiliza un quantum fijo.

Si la ráfaga es mayor que 100 entra en el else de la condición y te envía un JOptionPain para dar a conocer que no es permitido una ráfaga mayor a 100 luego limpia el TextField JTFCapturaRafaga y se vuelve a enfocar en el mismo.

Botón Iniciar:

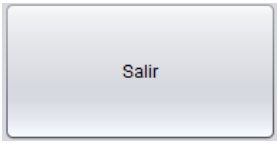

```
private void jBIniciarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    new Thread(new Hilo()).start(); //Crea un nuevo hilo
    Iniciar();
  }
```

Método jBIniciarActionPerformed (botón): crea un nuevo hilo de la clase hilo se inicia con el start() y luego se manda a llamar al método iniciar().

Constructores por defecto:

```
    private void jTFCapturaQuantumActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jLCantidadProcesosActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jLCantidadTiempoActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jLNumeroProcesoActionPerformed(java.awt.event.ActionEvent evt) {

        // TODO add your handling code here:
    }

    private void jLPorcentajeProcesoActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jTFCapturaPrioridadActionPerformed(java.awt.event.ActionEvent evt) {

    }


private void jTFCapturaIDActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jTFCapturaRafagaActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
```

Botón salir: Se utiliza para salir de la ejecución del programa




```
private void jBSalirActionPerformed(java.awt.event.ActionEvent evt) {
      System.exit(0);
   }
```

Clase main: Clase principal main que invoca a la clase procesar.

```
public static void main(String args[]) {
  /* Set the Nimbus look and feel */
/* Create and display the form */
      java.awt.EventQueue.invokeLater(new Runnable() {
         public void run() {
            new Procesar().setVisible(true);
         }
      });
   }
```

Hilo: Para conseguir que Java haga varias cosas a la vez o que el programa no se quede parado mientras realiza una tarea compleja, tenemos los hilos (Threads).

Para crear un hilo en java basta con heredar de la clase **Thread** y definir el método **run()**. Luego se instancia esta clase y se llama al método **start()** para que arranque el **hilo**.

Utilizamos el hilo para el inicio, ejecución, bloqueo y finalización de los procesos.

A continuación, se muestra el código con sus respectivos comentarios acerca de cada Instrucccion que se va realizando.

```
private class Hilo implements Runnable{ //Objeto de tipo Hilo con extension ejectubale
   @Override

 public void run(){
      int estado=1; //Estado de while que indica si se puede seguir o no
      int i=0; // contador de while

      while(estado!=0){
         while(i<Contador){ //Recorrer las filas
            Cargar(i);
            if(ResiduoRafaga!=0 && ResiduoRafaga>Quantum){ //Ejecutando Procesos

               if(ResiduoRafaga!=13 || ResiduoRafaga!=27){
                  for(int c=1; c<=Quantum; c++){
                     if(ResiduoRafaga==13){
                        jTIngreso.setValueAt("BLOQUEO (3)",i,4);
                        ResiduoRafaga--;
                        Barra(Rafaga,ResiduoRafaga);
                        Pintar();
                        jTIngreso.setValueAt(String.valueOf(ResiduoRafaga),i,3);
                        TiempoProceso++;
                        break;
                     }
                     if(ResiduoRafaga==27){
                        jTIngreso.setValueAt("BLOQUEO (5)",i,4);
                        ResiduoRafaga--;
                        Barra(Rafaga,ResiduoRafaga);
                        Pintar();
                        jTIngreso.setValueAt(String.valueOf(ResiduoRafaga),i,3);
                        TiempoProceso++;
                        break;
                     }
                     else{
                        jTIngreso.setValueAt("Ejecutando",i,4);
                        ResiduoRafaga--;
                        Barra(Rafaga,ResiduoRafaga);
                        Pintar();
                        jTIngreso.setValueAt(String.valueOf(ResiduoRafaga),i,3);
                        TiempoProceso++;
                        Reposo();
                     }
                     jTIngreso.setValueAt("Espera",i,4);
                  }
```

```
if(ResiduoRafaga==0){
                jTIngreso.setValueAt("Terminado",i,4);
                Pintar();
                Informe(i);
                Borrar(i);



   jPBEstado.setValue(0);
            }
          }

        }else{

          if(ResiduoRafaga>0 && Quantum!=0){
            while(ResiduoRafaga>0){
              jTIngreso.setValueAt("Ejecutando",i,4);
              ResiduoRafaga--;
              Barra(Rafaga,ResiduoRafaga);
              Pintar();
              jTIngreso.setValueAt(String.valueOf(ResiduoRafaga),i,3);
              TiempoProceso++;
              Reposo();
            }
            jTIngreso.setValueAt("Espera",i,4);


            if(ResiduoRafaga==0 && Quantum!=0){
              jTIngreso.setValueAt("Terminado",i,4);
              Pintar();
              Informe(i);
              Borrar(i);
              jPBEstado.setValue(0);
             }

            if(ResiduoRafaga==13){
              jTIngreso.setValueAt("BLOQUEO (3)",i,4);
              ResiduoRafaga--;
              Barra(Rafaga,ResiduoRafaga);
              Pintar();
              jTIngreso.setValueAt(String.valueOf(ResiduoRafaga),i,3);
              TiempoProceso++;
            }
```

```
if(ResiduoRafaga==27){
                jTIngreso.setValueAt("BLOQUEO (5)",i,4);
                ResiduoRafaga--;
                Barra(Rafaga,ResiduoRafaga);
                Pintar();
                jTIngreso.setValueAt(String.valueOf(ResiduoRafaga),i,3);
                TiempoProceso++;
            }



            }else{
                if(ResiduoRafaga==0 && Quantum!=0){
                    jTIngreso.setValueAt("Terminado",i,4);
                    Pintar();
                    Informe(i);
                    Borrar(i);
                    jPBEstado.setValue(0);
                }
            }
        }
        jLNumeroProceso.setText(String.valueOf("")); //Borrar contenido
        jLPorcentajeProceso.setText(String.valueOf(""));
        i++;
    }
    i=0;
    jLNumeroProceso.setText(String.valueOf("")); //Borrar contenido
    jLPorcentajeProceso.setText(String.valueOf(""));

  }

 }
}
```

Método reposo(): consiste en pausar en fracciones de tiempo para que percatemos lo que va sucediendo en el programa, se encuentra en milisegundos, se utiliza el método sleep () de la clase hilo de java, la cual hace una interrupción entre caga proceso en este caso en 600 milisegundos.

```
public void Reposo(){
   try{
      Thread.sleep(600); //Dormir sistema
   }catch(InterruptedException ex){
      Logger.getLogger(Procesar.class.getName()).log(Level.SEVERE,null,ex);
   }

}
```

Método cargar(): Recibe n valor entero, se utiliza para cargar los valores de la tabla por fila dentro de las variables y luego con una condicional se carga en el Label jLNumeroProceso el valor de la variable NProceso la cual representa el número del proceso cuando fue ingresado.

```
public void Cargar(int i){ //Carga los valores de la Tabla
   NProceso=(int)jTIngreso.getValueAt(i,0);
   Rafaga=parseInt((String)(jTIngreso.getValueAt(i,1)));
   Quantum=parseInt((String)(jTIngreso.getValueAt(i,2)));
   ResiduoRafaga=parseInt((String)(jTIngreso.getValueAt(i,3)));
   IdProceso=parseInt((String)(jTIngreso.getValueAt(i,5)));
   IdPrioridad=parseInt((String)(jTIngreso.getValueAt(i,6)));
   if(NProceso>0){
      jLNumeroProceso.setText(String.valueOf(NProceso));
   }

}
```

Informe: Consiste en agregar en la tabla los procesos terminados, recibe un valor entero el cual es la posición de tabla, se crea un modelo por defecto de la tabla jTFinal, luego se crea un objeto miTabla con el número de casillas de acuerdo a los campos de dicho modelo de la tabla jTFinal se ingresan los datos de las variables que ya hemos procesado (Ráfaga, quantum, TiempoProsesado), el string del estado que en este caso cambia a terminado y el contador +1 el cual representa el número de proceso, luego al modelo se le agrega los valores de miTabla y luego a la tabla jTFinal se le agrega los valores de modelo, luego se lleva el conteo de los procesos que finalizan con la variable CantidadProcesos la cual aumenta por cada proceso terminado en 1 y por ultimo imprime en los label jLCantidadProcesos el valor de dicha variable CantidadProcesos la cual nos muestra el número de proceso terminados y al label jLCantidadTiempo el valor de la variable TiempoProceso el cual representa el tiempo final en que se terminaron todos los procesos.

```
public void Informe(int c){
   DefaultTableModel modelo2 = (DefaultTableModel) jTFinal.getModel();

   Object[] miTabla= new Object[6];
   miTabla[0]= c+1;
   miTabla[1]= IdProceso;
   miTabla[2]= Rafaga;
   miTabla[3]= Quantum;
   miTabla[4]= TiempoProceso+" Segundos";
   miTabla[5]= "Terminado";


   modelo2.addRow(miTabla);
   jTFinal.setModel(modelo2);
   CantidadProcesos++;
   jLCantidadProcesos.setText(String.valueOf(CantidadProcesos+" Terminados"));
   jLCantidadTiempo.setText(String.valueOf(TiempoProceso+" Segundos"));
}
```

Borrar: Consiste en borrar los procesos de la tabla jTIngreso que cambien a estado terminado, recibe un valor entero el cual representa el número de la fila que se está ejecutando en dicho momento y cambia dichos valores por 0 y en caso del estado por asteriscos.

```
public void Borrar(int c){ //Elimina los registros de la tabla procesos
   jTIngreso.setValueAt(0,c,0);
   jTIngreso.setValueAt("0",c,1);
   jTIngreso.setValueAt("0",c,2);
   jTIngreso.setValueAt("0",c,3);
   jTIngreso.setValueAt("Finalizado",c,4);
}
```

Barra: recibe 2 valores enteros utilizando variables locales (valor, ráfaga, porcentaje) para obtener los valores luego realiza la aritmética del porcentaje y lo guarda dicho valor en la variable ValorBarra y seguido lo imprime en el label jLProcentajeProceso.

```
public void Barra(int rafaga, int residuo){ //Calcula porcentaje de la barra y su progreso
      int Rafaga=rafaga;
      int valor=100/rafaga;
      int porcentaje=100-(valor*residuo);
      ValorBarra=porcentaje;
      jLPorcentajeProceso.setText(String.valueOf(ValorBarra+"%"));
}
```

Pintar: Nos indica en tiempo real el porcentaje del progreso del proceso que se está ejecutando, enviando el dato de la variable ValorBarra a la barra jPBEstado y luego ejecuta la función repaint () para que nos muestre constantemente el progreso en la barra.

```
 public void Pintar(){
      jPBEstado.setValue(ValorBarra);
      jPBEstado.repaint();
   }
```

8. Diccionario de variables.

Contador: se utiliza para manejar el número de cada proceso y la cantidad de procesos ingresados.

NProceso: representa el número de proceso y para imprimir dicho valor en el label jLNumeroProceso.

IdProceso: representa en id único de cada proceso y se utiliza para imprimir el valor en la tabla jTIngreso

Ráfaga: para la obtención y manejo de los datos de la ráfaga inicial de cada proceso por individual
con el fin de ingresarlos a la tabla jTIngreso, procesarlos y para obtener el porcentaje de barra.
(Representa el tiempo necesario que debe utilizar en procesador para finalizar).

Quantum: representa el valor del quantum el cual en este caso es fijo y solo tendrá un valor desde
el momento del ingreso y se utiliza para imprimir en la tabla jTIngreso y para el momento de procesar las ráfagas y los procesos.

ResiduoRáfaga: se utiliza para contener el valor de la diferencia de ráfaga (tiempo necesario del uso del procesador para terminar) y el quantum (tiempo límite que un proceso puede utilizar el procesador),
con el fin de procesarlos, cambiar los estados del proceso y obtener el porcentaje de barra.

ValorBarra: se utiliza para guardar el valor aritmético que representa el porcentaje.