

* David Alexander Cardenas Almendares 20121003387
* Cristian Alexander Martínez Ochoa 20131015700
* Ariel Isaí Turcios García 20131006640
* Franklin Romario Chavarria Lainez 20131007637
* Emerson Joel Amador 20121016684

INTEGRANTES:

**Proyecto Sistemas Operativos I**

**Simulador de Gestor de Procesos**

**PERIODO 1 2020**



Manual Técnico

**Índice**

1. Objetivos ………………………………………………………………………………….. 3

1.1 Objetivos específicos …………………………………………………………. 3

1. Alcance …………………………………………………………………………………… 3
2. Requerimientos técnicos ……………………………………………………….. 4

3.1 Software ………………………………………………………………………….. 4

3.2 Hardware ………………………………………………………………………… 4

1. Herramientas utilizadas para el desarrollo …………………………… 5
2. Instalación ……………………………………………………………………………. 5
3. Configuración ………………………………………………………………………… 6
4. Diccionario de clases y métodos ………………………………………………………………. 6 -

**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

.

.

.

}

**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.

/\*\*

\* 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.



**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.

**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(""));

}

}

}