Catedra: Sistemas Operativos
Universidad: UMG
Repositorio de GITHUB
Alumno: 7691-19-11984 Victor Hugo Guerra Rodríguez
7691-15-9407 Edgar Adolfo Telles Veliz
https://github.com/v1ct0r-gu3rr4/sistemasOperativos.git

CLASE CONTROLLER

```
package controller;


import java.awt.Font;

import java.awt.Label;

import java.awt.event.KeyEvent;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.Map;

import java.util.Queue;

import java.util.Random;

import javax.swing.Icon;

import javax.swing.ImageIcon;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

import model.Marco;

import model.Pagina;

import model.Proceso;

import view.ConfigSO;

import view.ControlPanel;


/**
```

Catedra: Sistemas Operativos Alumno: 7691-19-11984 Victor Hugo Guerra Rodríguez
Universidad: UMG 7691-15-9407 Edgar Adolfo Telles Veliz
Repositorio de GITHUB https://github.com/v1ct0r-gu3rr4/sistemasOperativos.git

```
 *
 * @author Edgar Telles


 */
public class Controller {


    private JLabel labelInfo;
    public static HashMap<Integer, Proceso> procesos;
    public static Queue<Proceso> colaMemoriaPrincipal;
    public static Queue<Proceso> colaProcesos;
    public static Marco[] memoriaSecundaria;
    public static Marco[] memoriaPrincipal;
    public static int cantMarcosOcupados;
    public static int cantEspaciosOcupadosMS;
    public static int tamañoPagina;
    public static int tamañoMemPrincipal;
    public static int tamañoMemSecundaria;
    public static boolean changing;
    private static ControlPanel controlPanel;


    public Controller() {
        this.labelInfo = new JLabel();
        this.procesos = new HashMap<>();
        this.colaMemoriaPrincipal = new LinkedList<>();
        this.colaProcesos = new LinkedList<>();
        this.cantMarcosOcupados = 0;
        this.cantEspaciosOcupadosMS = 0;
    }


    public void initSO() {
```

```
        changing = false;
        ConfigSO configso = new ConfigSO(this);
        configso.setVisible(true);
    }

    public void closeFrame(JFrame frame){
        try {
            int dialogButton = JOptionPane.YES_NO_OPTION;
            int result = JOptionPane.showConfirmDialog(null, "¿Edgar o VIctor deseas salir?", "Bye", dialogButton);
            if (result == 0) {
                System.exit(0);
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(frame, e);
        }
    }

    public void minimizeFrame(JFrame frame){
        frame.setState(frame.ICONIFIED);
    }

    public void mostrarInfo(JFrame frame) {
        Icon icono = new ImageIcon(getClass().getResource(""));
        JOptionPane.showMessageDialog(frame, "Edgar Telles y Victor Guerra \n "
                + "    Universida mariano galvez de Guatemala - Ing. SISTEMAS\n "
                + "                    5 de noviembre del 2022", "", JOptionPane.INFORMATION_MESSAGE, icono);
    }

    public void validarSoloNumeros(KeyEvent evt, JFrame frame) {
        char validar = evt.getKeyChar();
```

```
        if (!Character.isDigit(validar)) {
            frame.getToolkit().beep();
            evt.consume();
        }
    }

    public void instalarSO(ConfigSO frame) {
        System.out.println("_  _ _    _          ____ ___          ______          _     _          \n" +
"Proyecto Final: VICTOR GUERRA Y EDGAR TELLES");
        int tamPrincipal = 0, tamSecundaria = 0, tamPaginas = 0;
        try {
            tamPrincipal = Integer.parseInt(frame.fieldTamPrincipal.getText());
            this.tamañoMemPrincipal = tamPrincipal;
            tamSecundaria = Integer.parseInt(frame.fieldTamSecundaria.getText());
            this.tamañoMemSecundaria = tamSecundaria;
            tamPaginas = Integer.parseInt(frame.fieldTamPaginas.getText());
            this.tamañoPagina = tamPaginas;
        } catch (Exception e) {
        }

        // Verificamos los campos ingresados
        if (frame.fieldTamPrincipal.getText() == "" || frame.fieldTamSecundaria.getText() == "" ||
frame.fieldTamPaginas.getText() == "" || tamPrincipal == 0 || tamPaginas == 0 || tamSecundaria == 0 || tamPaginas >
tamPrincipal) {
            JOptionPane.showMessageDialog(frame, "Introduzca bien los campos. No pueden haber ceros, el tamaño de la
página no puede ser mayor al tamaño de la memoria principal.");
            return;
        }
        // Verificamos que el valor del tamaño de la página entre exacto en la memoria principal y en la secundaria
        if (!(this.tamañoMemPrincipal % this.tamañoPagina == 0) || !(this.tamañoMemSecundaria % this.tamañoPagina ==
0)) {
```

```
        JOptionPane.showMessageDialog(frame, "El tamaño de la página tiene que entrar n número de veces de manera exacta en las memorias");
        return;
    }

    // Inicializamos las memorias
    this.memoriaPrincipal = new Marco[tamPrincipal / tamPaginas];
    this.memoriaSecundaria = new Marco[tamSecundaria / tamPaginas];

    // Los campos fueron introducidos de manera exitosa, se abre el Panel de Control
    ControlPanel controlP = new ControlPanel(this, tamPrincipal, tamSecundaria, tamPaginas);
    frame.dispose();
    controlP.setVisible(true);
    this.initTables(controlP);
    this.controlPanel = controlP;
  }

  private void initTables(ControlPanel controlP) {
    // Tamaño de cada columna
    controlP.tablePrincipal.getTableHeader().setReorderingAllowed(false);
    controlP.tablePrincipal.getTableHeader().setResizingAllowed(false);
    controlP.tablePrincipal.getColumnModel().getColumn(0).setPreferredWidth(65);
    controlP.tablePrincipal.getColumnModel().getColumn(1).setPreferredWidth(50);
    controlP.tablePrincipal.getColumnModel().getColumn(2).setPreferredWidth(70);
    controlP.tablePrincipal.getColumnModel().getColumn(3).setPreferredWidth(110);
    controlP.tablePrincipal.getColumnModel().getColumn(4).setPreferredWidth(40);
    controlP.tablePrincipal.getColumnModel().getColumn(5).setPreferredWidth(92);
    controlP.tablePrincipal.getColumnModel().getColumn(6).setPreferredWidth(96);
    // Altura de cada renglón
    controlP.tablePrincipal.setRowHeight(20);
```

```
        // Tamaño de cada columna
        controlP.tableSecundaria.getTableHeader().setReorderingAllowed(false);
        controlP.tableSecundaria.getTableHeader().setResizingAllowed(false);
        controlP.tableSecundaria.getColumnModel().getColumn(0).setPreferredWidth(130);
        controlP.tableSecundaria.getColumnModel().getColumn(1).setPreferredWidth(115);
        controlP.tableSecundaria.getColumnModel().getColumn(2).setPreferredWidth(150);
        controlP.tableSecundaria.getColumnModel().getColumn(3).setPreferredWidth(110);
        // Altura de cada renglón
        controlP.tableSecundaria.setRowHeight(20);

        // Tamaño de cada columna
        controlP.tableLista.getTableHeader().setReorderingAllowed(false);
        controlP.tableLista.getTableHeader().setResizingAllowed(false);
        controlP.tableLista.getColumnModel().getColumn(0).setPreferredWidth(50);
        controlP.tableLista.getColumnModel().getColumn(1).setPreferredWidth(130);
        controlP.tableLista.getColumnModel().getColumn(2).setPreferredWidth(115);
        controlP.tableLista.getColumnModel().getColumn(3).setPreferredWidth(100);
        controlP.tableLista.getColumnModel().getColumn(4).setPreferredWidth(180);
        controlP.tableLista.getColumnModel().getColumn(5).setPreferredWidth(60);
        controlP.tableLista.getColumnModel().getColumn(6).setPreferredWidth(60);
        // Altura de cada renglón
        controlP.tableLista.setRowHeight(20);
    }

    public void clearTableSelection(JTable table1, JTable table2, JTable table3) {
        table1.clearSelection();
        table2.clearSelection();
        table3.clearSelection();
    }
```

```
public void initDatosDelPanelDeControl(ControlPanel controlP) {
    DefaultTableModel modeloPrincipal = (DefaultTableModel) controlP.tablePrincipal.getModel();
    DefaultTableModel modeloSecundaria = (DefaultTableModel) controlP.tableSecundaria.getModel();
    DefaultTableModel modeloProcesos = (DefaultTableModel) controlP.tableLista.getModel();

    // Iniciamos la tabla de memoria principal
    int numMarcosPrincipal = (this.tamañoMemPrincipal / this.tamañoPagina);
    for (int i = 0; i < numMarcosPrincipal; i++) {
        Marco marco = new Marco("0x" + Integer.toHexString(i * this.tamañoPagina), i + 1, null, true);
        // Agregamos el marco a memoria principal
        this.memoriaPrincipal[i] = marco;
        modeloPrincipal.addRow(new Object[]{
            marco.getDirFisica(), marco.getNumMarco()
        });
    }

    // Iniciamos la tabla de memoria secundaria
    int numEspaciosSecundaria = (this.tamañoMemSecundaria / this.tamañoPagina);
    for (int i = 0; i < numEspaciosSecundaria; i++) {
        Marco marco = new Marco(String.valueOf(i + 1), i + 1, null, false);
        // Agregamos el marco a memoria secundaria
        this.memoriaSecundaria[i] = marco;
        modeloSecundaria.addRow(new Object[]{
            i + 1
        });
    }

    // Iniciamos la tabla de procesos
    controlP.jPanel1.setVisible(false);
    this.labelInfo.setFont(new Font("Calibrar", 1, 36));
```

```
        this.labelInfo.setText("No hay procesos creados todavía..");
        this.labelInfo.setBounds(550, 120, 600, 80);
        this.labelInfo.setVisible(true);
        controlP.panelContenido.add(this.labelInfo);

        DecimalFormat formatea = new DecimalFormat("###,###.##");

        // Iniciamos estadísticas
        controlP.labelProcesosCreados.setText("0");
        controlP.labelMarcosPagina.setText(formatea.format(numMarcosPrincipal));
        controlP.labelTamañoPagina.setText(formatea.format(this.tamañoPagina));
        controlP.labelEspaciosSecundaria.setText(formatea.format(numEspaciosSecundaria));

        // Iniciamos labels de tablas
        controlP.labelTamTotalPrincipal.setText(formatea.format(this.tamañoMemPrincipal));
        controlP.labelTamUsadaPrincipal.setText("0");
        controlP.labelTamDisponiblePrincipal.setText(formatea.format(this.tamañoMemPrincipal));
        controlP.labelTamTotalSecundaria.setText(formatea.format(this.tamañoMemSecundaria));
        controlP.labelTamUsadaSecundaria.setText("0");
        controlP.labelTamDisponibleSecundaria.setText(formatea.format(this.tamañoMemSecundaria));
        controlP.progressPrincipal.setValue(0);
        controlP.progressSecundaria.setValue(0);

        // Iniciamos los textFields de crearProcesos
        controlP.fieldNombreProceso.setText("Proceso #1");
        controlP.fieldTamañoProceso.setText("0");
    }

    private void aparecerTablaProcesos(ControlPanel controlP) {
        controlP.jPanel1.setVisible(true);
```

```
        this.labelInfo.setVisible(false);

    }


    public void crearProceso(ControlPanel controlP) {

        if(controlP.fieldNombreProceso.getText().equals("") || controlP.fieldTamañoProceso.getText().equals("")) {

            JOptionPane.showMessageDialog(controlP, "Ingrese datos correctamente", "BEEEP ERROR ",
JOptionPane.ERROR_MESSAGE);

            return;

        }


        if( (Double.parseDouble(controlP.fieldTamañoProceso.getText()) / (double)tamañoPagina ) / 2 >
memoriaPrincipal.length || Integer.parseInt(controlP.fieldTamañoProceso.getText()) == 0) {

            JOptionPane.showMessageDialog(controlP, "La mitad del tamaño del proceso no puede ser más grande que la
cantidad de espacios de la Memoria Principal", "Error", JOptionPane.ERROR_MESSAGE);

            return;

        }

        this.aparecerTablaProcesos(controlP);

        // Obtenemos el nombre y el tamaño de los textFields

        String nombreProceso = controlP.fieldNombreProceso.getText();

        int tamañoProceso = Integer.parseInt(controlP.fieldTamañoProceso.getText());

        // Generamos el random del tiempo que se va a ejecutar

        Random random = new Random();

        int tiempo = random.nextInt(5) + 1;

        // Agregamos el proceso al HashMap

        Proceso procesoNuevo = new Proceso(this.procesos.size(), nombreProceso, (double) tiempo, tamañoProceso);

        this.procesos.put(this.procesos.size(), procesoNuevo);

        // Agregar proceso a la cola

        this.putFirst(procesoNuevo);

        procesoNuevo.startTimer();

        // Aumentamos el número de procesos creados

        controlP.labelProcesosCreados.setText(String.valueOf(Integer.parseInt(controlP.labelProcesosCreados.getText()) +
1));
```

```
        // Llenamos los textFields de crear procesos
        controlP.fieldNombreProceso.setText("Proceso #" + (this.procesos.size() + 1));
        controlP.fieldTamañoProceso.setText("0");

        // Meter en la memoria principal para que se ejecute
        // Si la mitad de las paginas del proceso entran en MP
        if (procesoNuevo.getMitad() <= this.memoriaPrincipal.length - this.cantMarcosOcupados) {
            this.crearSiMitad(procesoNuevo);
        } else {
            // Si hay que reemplazar en MP, se saca de la memoria principal y se mete el nuevo proceso
            this.crearConReemplazo(procesoNuevo);
        }

        // Agregamos el proceso a la tabla de procesos
        DecimalFormat formatea = new DecimalFormat("###,###.##");
        DefaultTableModel modelo = (DefaultTableModel) controlP.tableLista.getModel();
        modelo.addRow(new Object[]{
            procesoNuevo.getID(), procesoNuevo.getNombre(), formatea.format(procesoNuevo.getTamaño()),
procesoNuevo.getCantPaginas(), procesoNuevo.getEstado(), procesoNuevo.getCantPagMP(),
procesoNuevo.getCantPagMS()
        });

    }

    public static void actualizarMemorias() {
        // Actualizamos MP
        for (int i = 0; i < controlPanel.tablePrincipal.getRowCount(); i++) {
            if (memoriaPrincipal[i].getPagina() != null) {
                int ID = memoriaPrincipal[i].getPagina().getIDProceso();
                Proceso proceso = procesos.get(ID);
                controlPanel.tablePrincipal.setValueAt(ID, i, 2);
```

```
            controlPanel.tablePrincipal.setValueAt(proceso.getNombre(), i, 3);
            controlPanel.tablePrincipal.setValueAt(memoriaPrincipal[i].getPagina().getNumPagina(), i, 4);
            controlPanel.tablePrincipal.setValueAt(memoriaPrincipal[i].getPagina().getTamaño(), i, 5);
            controlPanel.tablePrincipal.setValueAt(memoriaPrincipal[i].getPagina().getTamañoFragmentacion(), i, 6);
        } else {
            controlPanel.tablePrincipal.setValueAt("", i, 2);
            controlPanel.tablePrincipal.setValueAt("", i, 3);
            controlPanel.tablePrincipal.setValueAt("", i, 4);
            controlPanel.tablePrincipal.setValueAt("", i, 5);
            controlPanel.tablePrincipal.setValueAt("", i, 6);
        }
    }
    // Actualizamos MS
    for (int i = 0; i < controlPanel.tableSecundaria.getRowCount(); i++) {
        if (memoriaSecundaria[i].getPagina() != null) {
            int ID = memoriaSecundaria[i].getPagina().getIDProceso();
            Proceso proceso = procesos.get(ID);
            controlPanel.tableSecundaria.setValueAt(ID, i, 1);
            controlPanel.tableSecundaria.setValueAt(proceso.getNombre(), i, 2);
            controlPanel.tableSecundaria.setValueAt(memoriaSecundaria[i].getPagina().getNumPagina(), i, 3);
        } else {
            controlPanel.tableSecundaria.setValueAt("", i, 1);
            controlPanel.tableSecundaria.setValueAt("", i, 2);
            controlPanel.tableSecundaria.setValueAt("", i, 3);
        }
    }

    // Actualizamos los tamaños de las memorias
    DecimalFormat formatea = new DecimalFormat("###,###.##");
    controlPanel.labelTamUsadaPrincipal.setText(formatea.format(cantMarcosOcupados * tamañoPagina));
```

```
        controlPanel.labelTamDisponiblePrincipal.setText(formatea.format(tamañoMemPrincipal - (cantMarcosOcupados * tamañoPagina)));

        controlPanel.labelTamUsadaSecundaria.setText(formatea.format(cantEspaciosOcupadosMS * tamañoPagina));

        controlPanel.labelTamDisponibleSecundaria.setText(formatea.format(tamañoMemSecundaria - (cantEspaciosOcupadosMS * tamañoPagina)));


        controlPanel.progressPrincipal.setValue(((cantMarcosOcupados * tamañoPagina) * 100) / tamañoMemPrincipal);

        controlPanel.progressSecundaria.setValue(((cantEspaciosOcupadosMS * tamañoPagina) * 100) / tamañoMemSecundaria);


        // Actualizamos la tabla de procesos

        for (int i = 0; i < controlPanel.tableLista.getRowCount(); i++) {

            if (procesos.get(i).getEstado().equals("BEEEP ERROR")) {

                controlPanel.tableLista.setValueAt("<html><font color=\"red\">" + procesos.get(i).getEstado() + "</font></html>", i, 4);

                controlPanel.tableLista.setValueAt(procesos.get(i).getCantPagMP(), i, 5);

                controlPanel.tableLista.setValueAt(procesos.get(i).getCantPagMS(), i, 6);

            } else {

                controlPanel.tableLista.setValueAt(procesos.get(i).getEstado(), i, 4);

                controlPanel.tableLista.setValueAt(procesos.get(i).getCantPagMP(), i, 5);

                controlPanel.tableLista.setValueAt(procesos.get(i).getCantPagMS(), i, 6);

            }

        }


    }


    public void setTimeout(Runnable runnable, int delay) {

        new Thread(() -> {

            try {

                Thread.sleep(delay);

                runnable.run();

            } catch (Exception e) {
```

```
            System.err.println(e);
        }
    }).start();
}

public void eliminarProceso(ControlPanel controlP) {
    // Verificamos si se seleccionó algun proceso
    if (controlP.tableLista.getSelectedRow() == -1) {
        JOptionPane.showMessageDialog(controlP, "Seleccione el proceso que desea eliminar", "BEEEP ERROR",
JOptionPane.ERROR_MESSAGE);
        return;
    }

    // Obtenemos el ID del proceso
    int ID = Integer.parseInt(String.valueOf(controlP.tableLista.getValueAt(controlP.tableLista.getSelectedRow(), 0)));
    Proceso proceso = this.procesos.get(ID);

    if (proceso.getEstado().equals("Eliminado")) {
        JOptionPane.showMessageDialog(controlP, "No se puede eliminar un proceso en Eliminado", "BEEEP ERROR",
JOptionPane.ERROR_MESSAGE);
        return;
    }

    if (!proceso.getEstado().equals("Eliminado")) {
        // Ponemos el tiempo máximo de ejecución
        proceso.setTiempoEjecucion(proceso.getTiempoMaxEjecucion());
    }
}

public static void eliminarProceso(Proceso proceso) {
    // Modificamos variables
```

```
        cantMarcosOcupados = cantMarcosOcupados - proceso.getCantPagMP();

        cantEspaciosOcupadosMS = cantEspaciosOcupadosMS - proceso.getCantPagMS();

        // Borramos el proceso de las memorias

        // MP

        for (int i = 0; i < memoriaPrincipal.length; i++) {

            // Encontramos el proceso en MP

            if (memoriaPrincipal[i].getPagina() != null) {

                if (memoriaPrincipal[i].getPagina().getIDProceso() == proceso.getID()) {

                    memoriaPrincipal[i].setPagina(null);

                }

            }

        }

        // MS

        for (int i = 0; i < memoriaSecundaria.length; i++) {

            // Encontramos el proceso en MS

            if (memoriaSecundaria[i].getPagina() != null) {

                if (memoriaSecundaria[i].getPagina().getIDProceso() == proceso.getID()) {

                    memoriaSecundaria[i].setPagina(null);

                }

            }

        }


        if (colaMemoriaPrincipal.contains(proceso)) {

            colaMemoriaPrincipal.remove(proceso);

        }


        actualizarMemorias();

    }


    public static void modificarProceso(int idProceso, boolean flag) {
```

```
        procesos.get(idProceso).modificarPagina(flag);

        Controller.actualizarMemorias();

    }


    public void crearSiMitad(Proceso procesoNuevo) {

        colaMemoriaPrincipal.offer(procesoNuevo);

        //System.out.println("Caben al menos la mitad de las páginas del " + procesoNuevo.getNombre());

        int marcosDispon = this.memoriaPrincipal.length - this.cantMarcosOcupados;


        int cantPagsMP = 0, cantPagsMS = 0;

        // Si no caben todas en MP

        if (marcosDispon - procesoNuevo.getPaginas().length < 0) {

            cantPagsMS = procesoNuevo.getPaginas().length - marcosDispon;

            cantPagsMP = procesoNuevo.getPaginas().length - cantPagsMS;

            // Aumentamos los marcos ocupados

            this.cantMarcosOcupados = this.cantMarcosOcupados + cantPagsMP;

            this.cantEspaciosOcupadosMS = this.cantEspaciosOcupadosMS + cantPagsMS;

            // Seteamos la cantidad de paginas en cada memoria

            procesoNuevo.setCantPagMP(cantPagsMP);

            procesoNuevo.setCantPagMS(cantPagsMS);

            // Agregamos al array de MP y MS

            int contPaginasPuestas = 0;

            // A la MP

            for (int i = 0; i < this.memoriaPrincipal.length; i++) {

                if (this.memoriaPrincipal[i].getPagina() == null && contPaginasPuestas < cantPagsMP) {

                    // Si encontramos el espacio vacío de la MP y aun tengo paginas que meter

                    this.memoriaPrincipal[i].setPagina(procesoNuevo.getPaginas()[contPaginasPuestas]);

                    this.memoriaPrincipal[i].getPagina().setIDProceso(procesoNuevo.getID());

                    this.memoriaPrincipal[i].getPagina().crearSetInMemoriaP(true);

                    contPaginasPuestas++;
```

```
            }
        }
        // A la MS
        int contAux = contPaginasPuestas;
        for (int i = 0; i < this.memoriaSecundaria.length; i++) {
            if (this.memoriaSecundaria[i].getPagina() == null && (procesoNuevo.getPaginas().length - contAux -
cantPagsMS) < cantPagsMS) {
                // Si encontramos el espacio vacío de la MS y aun tengo paginas que meter
                this.memoriaSecundaria[i].setPagina(procesoNuevo.getPaginas()[contPaginasPuestas]);
                this.memoriaSecundaria[i].getPagina().setIDProceso(procesoNuevo.getID());
                this.memoriaSecundaria[i].getPagina().crearSetInMemoriaP(false);
                contPaginasPuestas++;
                contAux--;
            }
        }

        this.actualizarMemorias();

    } else {
        // Caben todas en MP
        cantPagsMS = 0;
        cantPagsMP = procesoNuevo.getPaginas().length;
        // Seteamos la cantidad de paginas en cada memoria
        procesoNuevo.setCantPagMP(cantPagsMP);
        procesoNuevo.setCantPagMS(cantPagsMS);
        // Aumentamos los marcos ocupados
        this.cantMarcosOcupados = this.cantMarcosOcupados + cantPagsMP;
        this.cantEspaciosOcupadosMS = this.cantEspaciosOcupadosMS + cantPagsMS;
        // Agregamos a la Memoria Principal
        int contPaginasPuestas = 0;
        // A la MP
```

```
        for (int i = 0; i < this.memoriaPrincipal.length; i++) {
            if (this.memoriaPrincipal[i].getPagina() == null && contPaginasPuestas < cantPagsMP) {
                // Si encontramos el espacio vacío de la MP y aun tengo paginas que meter
                this.memoriaPrincipal[i].setPagina(procesoNuevo.getPaginas()[contPaginasPuestas]);
                this.memoriaPrincipal[i].getPagina().setIDProceso(procesoNuevo.getID());
                this.memoriaPrincipal[i].getPagina().crearSetInMemoriaP(true);
                contPaginasPuestas++;
            }
        }
        this.actualizarMemorias();
    }
}

public void crearConReemplazo(Proceso procesoNuevo) {
    int marcosDispon = this.memoriaPrincipal.length - this.cantMarcosOcupados;
    int cantPagsNecesitoEnMP = procesoNuevo.getMitad();
    int cantPagsNecesitoEnMS = procesoNuevo.getPaginas().length - cantPagsNecesitoEnMP;

    // Como los valores de arriba siempre se van a cumplir, actualizamos los valores
    procesoNuevo.setCantPagMP(cantPagsNecesitoEnMP);
    procesoNuevo.setCantPagMS(cantPagsNecesitoEnMS);
    actualizarMemorias();
    // Verificamos si hay espacio en memoria virtual para meter el proceso
    if (procesoNuevo.getPaginas().length > (marcosDispon + (memoriaSecundaria.length - cantEspaciosOcupadosMS))) {
        JOptionPane.showMessageDialog(null, "No hay espacio para crear ese proceso.");
        return;
    }
    // Se agrega el proceso a la cola de MP
    colaMemoriaPrincipal.offer(procesoNuevo);
    // Se verifica en la cola de MP los procesos y se ve cuantas páginas de cada uno de los procesos hay
```

```
    // que quitar para que entren las que necesito en MP
    // Verificar si hay espacios vacios en MP y meter las páginas que se puedan
    int numPagg = 0;
    // Si hay espacio en MP
    if (cantMarcosOcupados < this.memoriaPrincipal.length) {
        int espaciosVacios = this.memoriaPrincipal.length - cantMarcosOcupados;
        int numPag = 0;
        for (numPag = 0; numPag < espaciosVacios; numPag++) {
            for (int j = 0; j < memoriaPrincipal.length; j++) {
                if (memoriaPrincipal[j].getPagina() == null) {
                    memoriaPrincipal[j].setPagina(procesoNuevo.getPaginas()[numPag]);
                    procesoNuevo.getPaginas()[numPag].crearSetInMemoriaP(true);
                    cantPagsNecesitoEnMP--;
                    cantMarcosOcupados++;
                    break;
                }
            }
        }
        numPagg = numPag;
    }

    // Verifico que procesos voy a quitar
    boolean ready = false;
    // Recorro todos los procesos que están en la cola de MP
    for (Proceso pro : colaMemoriaPrincipal) {
        // Verifico si ya se pusieron todas las páginas en MP
        if (ready) {
            break;
        }
        // Recorro todas las páginas de el proceso
```

```
        for (int i = 0; i < pro.getPaginas().length; i++) {

            // Si la página está en MP la saco y meto la del proceso nuevo

            if (pro.getPaginas()[i].isInMemoriaPrincipal()) {

                if (cantPagsNecesitoEnMP <= 0) {

                    ready = true;

                    break;

                }

                cantPagsNecesitoEnMP--;

                pro.getPaginas()[i].setInMemoriaPrincipal(false);

                // Meto la página en MS y la saco de MP

                for (int j = 0; j < memoriaSecundaria.length; j++) {

                    if (memoriaSecundaria[j].getPagina() == null) {

                        memoriaSecundaria[j].setPagina(pro.getPaginas()[i]);

                        cantEspaciosOcupadosMS++;

                        break;

                    }

                }


                for (int j = 0; j < memoriaPrincipal.length; j++) {

                    // Si son del mismo proceso y el mismo numero de pagina

                    // System.out.println("Estoy en el " + pro.getNombre() + " y voy a sacar a la pag " +
pro.getPaginas()[i].getNumPagina());

                    if (memoriaPrincipal[j].getPagina().getNumPagina() == pro.getPaginas()[i].getNumPagina() &&
memoriaPrincipal[j].getPagina().getIDProceso() == pro.getPaginas()[i].getIDProceso()) {

                        memoriaPrincipal[j].setPagina(procesoNuevo.getPaginas()[numPagg]);

                        procesoNuevo.getPaginas()[numPagg].crearSetInMemoriaP(true);

                        numPagg++;

                        actualizarMemorias();

                    }

                }
```

```
            }
        }
    }

    for (int i = 0; i < memoriaSecundaria.length; i++) {
        if (memoriaSecundaria[i].getPagina() == null && cantPagsNecesitoEnMS > 0) {
            memoriaSecundaria[i].setPagina(procesoNuevo.getPaginas()[numPagg]);
            procesoNuevo.getPaginas()[numPagg - 1].crearSetInMemoriaP(false);
            cantPagsNecesitoEnMS--;
            numPagg++;
            cantEspaciosOcupadosMS++;
        }
        if (cantPagsNecesitoEnMS == 0) {
            break;
        }
    }
    actualizarMemorias();
}

public void putFirst(Proceso nuevo) {
    changing = true;
    Queue<Proceso> colaAux = new LinkedList<>();

    colaAux.offer(nuevo);

    while (!colaProcesos.isEmpty()) {
        Proceso proceso = colaProcesos.peek();
        if (proceso.getEstado().equals("Ejecución")) {
            if (proceso.isSuspendido()) {
                proceso.setEstado("Suspendido/Listo");
```

```
            } else {
                proceso.setEstado("Listo");
            }
        }
        colaAux.offer(colaProcesos.poll());
    }

    colaProcesos = colaAux;
    changing = false;
  }

  public void bloquearProceso(ControlPanel controlP) {
    // Verificamos si se seleccionó algun proceso
    if (controlP.tableLista.getSelectedRow() == -1) {
        JOptionPane.showMessageDialog(controlP, "Seleccione el proceso que desea bloquear/desbloquear", "BEEEP ERROR", JOptionPane.ERROR_MESSAGE);
        return;
    }

    // Obtenemos el ID del proceso
    int ID = Integer.parseInt(String.valueOf(controlP.tableLista.getValueAt(controlP.tableLista.getSelectedRow(), 0)));
    Proceso proceso = this.procesos.get(ID);

    if (proceso.getEstado().equals("Eliminado")) {
        JOptionPane.showMessageDialog(controlP, "No se puede bloquear un proceso en Eliminado", "BEEEP ERROR", JOptionPane.ERROR_MESSAGE);
        return;
    }

    changing = true;
    if (proceso.getEstado().endsWith("Listo") || proceso.getEstado().equals("Ejecución") ) {
```

```
            colaProcesos.remove(proceso);
        } else if (proceso.getEstado().endsWith("Bloqueado")) {
            putFirst(proceso);
        }

        if (proceso.getEstado().equals("Pausa")) {
            proceso.setEstado("Listo");
        } else if (proceso.getEstado().equals("Pausa/Bloqueado")) {
            proceso.setEstado("Pausa/Listo");
        } else if (proceso.getEstado().equals("Pausa/Listo")) {
            proceso.setEstado("Pausa/Bloqueado");
        } else if (proceso.getEstado().equals("Listo") || proceso.getEstado().equals("Ejecución") ) {
            proceso.setEstado("Bloqueado");
        }
        changing = false;
        actualizarMemorias();
    }

    public void suspenderProceso(ControlPanel controlP) {
        // Verificamos si se seleccionó algun proceso
        if (controlP.tableLista.getSelectedRow() == -1) {
            JOptionPane.showMessageDialog(controlP, "Seleccione el proceso que desea suspender/recuperar ", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Obtenemos el ID del proceso
        int ID = Integer.parseInt(String.valueOf(controlP.tableLista.getValueAt(controlP.tableLista.getSelectedRow(), 0)));
        Proceso proceso = this.procesos.get(ID);

        if (proceso.getEstado().equals("Eliminado")) {
```

```
        JOptionPane.showMessageDialog(controlP, "No se puede suspender un proceso en Eliminado", "BEEEP ERROR",
JOptionPane.ERROR_MESSAGE);
        return;
      }
      changing = true;

      if (proceso.getEstado().equals("Listo") || proceso.getEstado().equals("Bloqueado") ||
proceso.getEstado().equals("Ejecución")) {
        if (memoriaSecundaria.length - cantEspaciosOcupadosMS < proceso.getCantPagMP()) {
          JOptionPane.showMessageDialog(controlP, "No se puede suspender porque no hay espacio e n MEMORIA
VIRTUAL", "Error", JOptionPane.ERROR_MESSAGE);
          return;
        }
        if (proceso.getEstado().equals("Listo") || proceso.getEstado().equals("Ejecución")) {
          proceso.setEstado("Pausa/Listo");
        } else {
          proceso.setEstado("Pausa/Bloqueado");
        }
        // Se suspenden las páginas de MP a MS

        for (int i = 0; i < memoriaPrincipal.length; i++) {
          if (memoriaPrincipal[i].getPagina() != null && memoriaPrincipal[i].getPagina().getIDProceso() ==
proceso.getID()) {
            for (int j = 0; j < memoriaSecundaria.length; j++) {
              if (memoriaSecundaria[j].getPagina() == null) {
                cantEspaciosOcupadosMS++;
                cantMarcosOcupados--;
                memoriaPrincipal[i].getPagina().setInMemoriaPrincipal(false);
                memoriaSecundaria[j].setPagina(memoriaPrincipal[i].getPagina());
                memoriaPrincipal[i].setPagina(null);
                break;
              }
```

```
                }
            }

            if (proceso.getCantPagMP() <= 0) {
                break;
            }
        }
    } else if (proceso.getEstado().equals("Suspendido/Bloqueado")) {
        proceso.setEstado("Bloqueado");

        this.reemplazarAlRecuperar(proceso);
        // Meter páginas en MP
    } else if (proceso.getEstado().equals("Suspendido/Listo")) {
        // Lo saco de la cola y lo pongo de primero
        colaProcesos.remove(proceso);
        putFirst(proceso);
    }
    changing = false;
    actualizarMemorias();
}

void reemplazarAlRecuperar(Proceso proceso) {
    int marcosDispon = Controller.memoriaPrincipal.length - Controller.cantMarcosOcupados;
    int cantPagsNecesitoEnMP = proceso.getMitad();
    int cantPagsNecesitoEnMS = proceso.getPaginas().length - cantPagsNecesitoEnMP;
    int pagsMetereEnMP = proceso.getMitad() - proceso.getCantPagMP();
    if (pagsMetereEnMP <= 0) {
        System.out.println("Erro de humano, perdon :( ");
    }
    // Si hay espacio para meter las páginas sin tener que reemplazar
```

Alumno: 7691-19-11984 Victor Hugo Guerra Rodríguez
7691-15-9407 Edgar Adolfo Telles Veliz


```
if (marcosDispon >= proceso.getCantPagMS()) {
    pagsMetereEnMP = proceso.getCantPagMS();
}

if (marcosDispon >= pagsMetereEnMP) {
    // Metemos las páginas en MP
    for (int i = 0; i < Controller.memoriaPrincipal.length; i++) {
        if (Controller.memoriaPrincipal[i].getPagina() == null) {
            for (int j = 0; j < proceso.getPaginas().length; j++) {
                if (!proceso.getPaginas()[j].isInMemoriaPrincipal()) {
                    Controller.memoriaPrincipal[i].setPagina(proceso.getPaginas()[j]);
                    proceso.getPaginas()[j].setInMemoriaPrincipal(true);
                    pagsMetereEnMP--;
                    // Modificamos variables
                    Controller.cantEspaciosOcupadosMS--;
                    Controller.cantMarcosOcupados++;
                    break;
                }
            }
        }
        if (pagsMetereEnMP <= 0) {
            break;
        }
    }
    // Las sacamos de la MS
    for (int j = 0; j < Controller.memoriaSecundaria.length; j++) {
        // Si se encuentra una página del proceso en MS
        if (Controller.memoriaSecundaria[j].getPagina() != null) {
            if (Controller.memoriaSecundaria[j].getPagina().getIDProceso() == proceso.getID()) {
```

```
                Controller.memoriaSecundaria[j].setPagina(null);
                Controller.actualizarMemorias();
            }
        }
    }
} else {
    // Si hay que reemplazar
    // Verificar si hay algun marco disponible, pero sin ser todos los que se necesitan
    if (marcosDispon > 0) {
        // Buscamos el marco disponible y metemos la página en el marco de MP
        ArrayList<Integer> numPags = new ArrayList<>();
        for (int i = 0; i < Controller.memoriaPrincipal.length; i++) {
            if (Controller.memoriaPrincipal[i].getPagina() == null) {
                for (int j = 0; j < proceso.getPaginas().length; j++) {
                    if (!proceso.getPaginas()[j].isInMemoriaPrincipal()) {
                        proceso.eliminarPagina(proceso.getPaginas()[j]);
                        Controller.memoriaPrincipal[i].setPagina(proceso.getPaginas()[j]);
                        proceso.getPaginas()[j].setInMemoriaPrincipal(true);
                        numPags.add(j);
                        pagsMetereEnMP--;
                        marcosDispon--;
                        // Modificamos variables
                        Controller.cantEspaciosOcupadosMS--;
                        Controller.cantMarcosOcupados++;
                        Controller.actualizarMemorias();
                        break;
                    }
                }
            }
            if (marcosDispon <= 0) {
```

```
                break;
            }
        }
    }
    // Se meten las páginas que faltan por meter en MP (o en su defecto, todas las que tengan que estar)
    // Verifico que procesos voy a quitar
    boolean ready = false;
    // Recorro todos los procesos que están en la cola de MP
    for (Proceso pro : Controller.colaMemoriaPrincipal) {
        if (pro.equals(proceso)) {
            continue;
        }
        // Verifico si ya se pusieron todas las páginas en MP
        if (ready) {
            break;
        }
        // Recorro todas las páginas del proceso
        for (int i = 0; i < pro.getPaginas().length; i++) {
            // Si la página está en MP la saco y meto la del proceso nuevo
            if (pro.getPaginas()[i].isInMemoriaPrincipal()) {
                // Meto la página en MS y la saco de MP
                for (int j = 0; j < Controller.memoriaSecundaria.length; j++) {
                    if (Controller.memoriaSecundaria[j].getPagina() == null) {
                        Controller.memoriaSecundaria[j].setPagina(pro.getPaginas()[i]);
                        pro.getPaginas()[i].setInMemoriaPrincipal(false);
                        Controller.actualizarMemorias();
                        break;
                    }
                }
                for (int j = 0; j < Controller.memoriaPrincipal.length; j++) {
```

```
                    // Si son del mismo proceso y el mismo numero de pagina

                    // System.out.println("Estoy en el " + pro.getNombre() + " y voy a sacar a la pag " +
pro.getPaginas()[i].getNumPagina());

                    if (Controller.memoriaPrincipal[j].getPagina().getNumPagina() == pro.getPaginas()[i].getNumPagina() &&
Controller.memoriaPrincipal[j].getPagina().getIDProceso() == pro.getPaginas()[i].getIDProceso()) {

                      for (int k = 0; k < proceso.getPaginas().length; k++) {

                        if (!proceso.getPaginas()[k].isInMemoriaPrincipal()) {

                          proceso.eliminarPagina(proceso.getPaginas()[k]);

                          Controller.memoriaPrincipal[j].setPagina(proceso.getPaginas()[k]);

                          proceso.getPaginas()[k].setInMemoriaPrincipal(true);

                          pagsMetereEnMP--;

                          Controller.actualizarMemorias();

                          break;

                        }

                      }

                    }

                    pagsMetereEnMP--;

                    if (pagsMetereEnMP <= 0) {

                      ready = true;

                      break;

                    }

                  }

                }

              }

            }

            Controller.actualizarMemorias();

          }

        }
```

Catedra: Sistemas Operativos
Universidad: UMG
Repositorio de GITHUB
Alumno: 7691-19-11984 Victor Hugo Guerra Rodríguez
7691-15-9407 Edgar Adolfo Telles Veliz
https://github.com/v1ct0r-gu3rr4/sistemasOperativos.git

CLASE PROCESO:

```
package model;


import controller.Controller;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.Timer;


/**
 *
 * @author Edgar Telles
 */
public class Proceso {


    private final double clockTime = 1.5;
    private int ID;
    private String nombre;
    private String estado;
    private int cantPaginas;
    private int cantPagMP;
    private int cantPagMS;
    private Double tiempoMaxEjecucion;
    private Double tiempoEjecucion;
    private int tamaño;
    private Pagina[] paginas;


    public Proceso(int ID, String nombre, Double tiempoMaxEjecucion, int tamaño) {
        this.ID = ID;
        this.nombre = nombre;
```

```
        this.tiempoMaxEjecucion = tiempoMaxEjecucion;

        this.tiempoEjecucion = 0.0;

        this.tamaño = tamaño;

        this.cantPaginas = (int) Math.ceil((float) this.tamaño / Controller.tamañoPagina);

        this.paginas = new Pagina[cantPaginas];

        this.cantPagMP = 0;

        this.cantPagMS = 0;


        // Creamos las páginas del proceso

        this.crearPaginas();

        //this.verPaginas();


        if (Controller.colaProcesos.isEmpty()) {

            this.estado = "Ejecución";

        } else {

            this.estado = "Nuevo";

        }



        // Creamos el timer para saber los ciclos de ejecución

        System.out.println("\nSoy " + this.nombre + " y tengo " + this.tiempoMaxEjecucion + " para ejecutarme. Deberia
aparecer " + this.tiempoMaxEjecucion / 0.5);

    }


    public void startTimer() {

        Proceso proceso = this;


        Timer timer = new Timer((int) (clockTime * 1000), new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent ae) {

                // Si me toca ejecutarme
```

```
        if (proceso.estado.equals("Ejecución") && !Controller.changing) {
            Controller.actualizarMemorias();
            // Se aumenta un ciclo del tiempo de Ejecución
            proceso.setTiempoEjecucion(proceso.getTiempoEjecucion() + 0.5);
            System.out.println("Se ejecuta el proceso " + proceso.getID());

            if (proceso.getTiempoEjecucion() >= proceso.getTiempoMaxEjecucion()) {
                // Eliminar Proceso
                Controller.colaProcesos.poll();
                Controller.eliminarProceso(proceso);
                proceso.setEstado("Eliminado");
                Controller.actualizarMemorias();
                proceso.setCantPagMP(0);
                proceso.setCantPagMS(0);
                System.out.println("Se terminó el tiempo de ejecución de " + proceso.getNombre());
                return;
            } else {
                // Se pone de último el primero de la cola, es decir, este proceso se encola
                Controller.colaProcesos.offer(Controller.colaProcesos.poll());
                if (Controller.colaProcesos.size() != 1) {
                    proceso.setEstado("Listo");
                }
            }
        } else if (!Controller.colaProcesos.isEmpty() && Controller.colaProcesos.peek().ID == proceso.ID) {
            if (proceso.getEstado().equals("Suspendido/Listo")) {
                // Hay que reemplazar del primero de MP
                proceso.reemplazar();
            }
            proceso.setEstado("Ejecución");
        }
```

```
            Controller.actualizarMemorias();
        }
    });
    timer.start();
}

private void crearPaginas() {
    // Número de páginas completas sin fragmentación interna
    int paginasCompletas = this.tamaño / Controller.tamañoPagina;
    // Número y tamaño de páginas incompletas con fragmentación interna
    int paginasIncompletas = 0;
    int tamañoPagIncompleta = 0;
    if (paginasCompletas != this.cantPaginas) { // Si hay paginas incompletas
        paginasIncompletas = 1;
        tamañoPagIncompleta = this.tamaño % Controller.tamañoPagina;
    }
    // Creamos las páginas completas
    for (int i = 0; i < paginasCompletas; i++) {
        this.paginas[i] = new Pagina(i + 1, Controller.tamañoPagina, this.ID);
    }
    // Creamos la página incompleta en la última posición en caso de haber páginsa incompletas
    if (paginasIncompletas == 1) {
        this.paginas[this.cantPaginas - 1] = new Pagina(this.cantPaginas, tamañoPagIncompleta, this.ID);
    }
}

public void reemplazar() {
    int marcosDispon = Controller.memoriaPrincipal.length - Controller.cantMarcosOcupados;
    int cantPagsNecesitoEnMP = this.getMitad();
    int cantPagsNecesitoEnMS = this.getPaginas().length - cantPagsNecesitoEnMP;
```

```
    int pagsMetereEnMP = this.getMitad() - this.getCantPagMP();

    if (pagsMetereEnMP <= 0) {

       System.out.println("Error, perdon :( ");

    }


    if (marcosDispon >= this.cantPagMS) {

       pagsMetereEnMP = this.getCantPagMS();

    }


    // Si hay espacio para meter las páginas sin tener que reemplazar

    if (marcosDispon >= pagsMetereEnMP) {

       // Metemos las páginas en MP

       for (int i = 0; i < Controller.memoriaPrincipal.length; i++) {

          if (Controller.memoriaPrincipal[i].getPagina() == null) {

             for (int j = 0; j < this.paginas.length; j++) {

                if (!this.paginas[j].isInMemoriaPrincipal()) {

                   Controller.memoriaPrincipal[i].setPagina(this.paginas[j]);

                   this.paginas[j].setInMemoriaPrincipal(true);

                   pagsMetereEnMP--;

                   // Modificamos variables

                   Controller.cantEspaciosOcupadosMS--;

                   Controller.cantMarcosOcupados++;

                   break;

                }

             }

          }

          if (pagsMetereEnMP <= 0) {

             break;

          }

       }
```

```
        // Las sacamos de la MS
        for (int j = 0; j < Controller.memoriaSecundaria.length; j++) {
            // Si se encuentra una página del proceso en MS
            if(Controller.memoriaSecundaria[j].getPagina() != null) {
                if (Controller.memoriaSecundaria[j].getPagina().getIDProceso() == this.getID()) {
                    Controller.memoriaSecundaria[j].setPagina(null);
                    Controller.actualizarMemorias();
                }
            }
        }
    } else {
        // Si hay que reemplazar
        // Verificar si hay algun marco disponible, pero sin ser todos los que se necesitan
        if (marcosDispon > 0) {
            // Buscamos el marco disponible y metemos la página en el marco de MP
            ArrayList<Integer> numPags = new ArrayList<>();
            for (int i = 0; i < Controller.memoriaPrincipal.length; i++) {
                if (Controller.memoriaPrincipal[i].getPagina() == null) {
                    for (int j = 0; j < this.paginas.length; j++) {
                        if (!this.paginas[j].isInMemoriaPrincipal()) {
                            this.eliminarPagina(this.paginas[j]);
                            Controller.memoriaPrincipal[i].setPagina(this.paginas[j]);
                            this.paginas[j].setInMemoriaPrincipal(true);
                            numPags.add(j);
                            pagsMetereEnMP--;
                            marcosDispon--;
                            // Modificamos variables
                            Controller.cantEspaciosOcupadosMS--;
                            Controller.cantMarcosOcupados++;
                            Controller.actualizarMemorias();
```

```
                    break;
                }
            }
        }
        if (marcosDispon <= 0) {
            break;
        }
    }
}
// Se meten las páginas que faltan por meter en MP (o en su defecto, todas las que tengan que estar)
// Verifico que procesos voy a quitar
boolean ready = false;
// Recorro todos los procesos que están en la cola de MP
for (Proceso pro : Controller.colaMemoriaPrincipal) {
    if(pro.equals(this)){
        continue;
    }
    // Verifico si ya se pusieron todas las páginas en MP
    if (ready) {
        break;
    }
    // Recorro todas las páginas del proceso
    for (int i = 0; i < pro.getPaginas().length; i++) {
        // Si la página está en MP la saco y meto la del proceso nuevo
        if (pro.getPaginas()[i].isInMemoriaPrincipal()) {
            // Meto la página en MS y la saco de MP
            for (int j = 0; j < Controller.memoriaSecundaria.length; j++) {
                if (Controller.memoriaSecundaria[j].getPagina() == null) {
                    Controller.memoriaSecundaria[j].setPagina(pro.getPaginas()[i]);
                    pro.getPaginas()[i].setInMemoriaPrincipal(false);
```

```
                    Controller.actualizarMemorias();
                    break;
                }
            }
            for (int j = 0; j < Controller.memoriaPrincipal.length; j++) {
                // Si son del mismo proceso y el mismo numero de pagina
                // System.out.println("Estoy en el " + pro.getNombre() + " y voy a sacar a la pag " +
pro.getPaginas()[i].getNumPagina());
                if (Controller.memoriaPrincipal[j].getPagina().getNumPagina() == pro.getPaginas()[i].getNumPagina() &&
Controller.memoriaPrincipal[j].getPagina().getIDProceso() == pro.getPaginas()[i].getIDProceso()) {
                    for (int k = 0; k < this.getPaginas().length; k++) {
                        if (!this.getPaginas()[k].isInMemoriaPrincipal()) {
                            this.eliminarPagina(this.getPaginas()[k]);
                            Controller.memoriaPrincipal[j].setPagina(this.paginas[k]);
                            this.paginas[k].setInMemoriaPrincipal(true);
                            pagsMetereEnMP--;
                            Controller.actualizarMemorias();
                            break;
                        }
                    }
                }
            }
            pagsMetereEnMP--;
            if (pagsMetereEnMP <= 0) {
                ready = true;
                break;
            }
        }
    }
}
```

```
        Controller.actualizarMemorias();

    }


    public void verPaginas() {

        System.out.println("\n PROCESO NO: " + this.nombre);

        System.out.println("----------------------------");

        for (int i = 0; i < this.paginas.length; i++) {

            System.out.println("PAGINA NO: " + this.paginas[i].getNumPagina() + " PESO MB " + this.paginas[i].getTamaño() +
" Frangmentado en " + this.paginas[i].getTamañoFragmentacion());

        }

        System.out.println("----------------------------");

    }


    public int getCantPaginas() {

        return cantPaginas;

    }


    public String getEstado() {

        return estado;

    }


    public int getID() {

        return ID;

    }


    public String getNombre() {

        return nombre;

    }


    public Double getTiempoEjecucion() {

        return tiempoEjecucion;
```

Alumno: 7691-19-11984 Victor Hugo Guerra Rodríguez
7691-15-9407 Edgar Adolfo Telles Veliz


```
    }

    public int getTamaño() {
        return tamaño;
    }

    public Pagina[] getPaginas() {
        return paginas;
    }

    public Double getTiempoMaxEjecucion() {
        return tiempoMaxEjecucion;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public void setEstado(String estado) {
        this.estado = estado;
    }

    public void setCantPaginas(int cantPaginas) {
        this.cantPaginas = cantPaginas;
    }
```

```
    public void setTiempoMaxEjecucion(Double tiempoMaxEjecucion) {
        this.tiempoMaxEjecucion = tiempoMaxEjecucion;
    }

    public void setTiempoEjecucion(Double tiempoEjecucion) {
        this.tiempoEjecucion = tiempoEjecucion;
    }

    public void setTamaño(int tamaño) {
        this.tamaño = tamaño;
    }

    public void setPaginas(Pagina[] paginas) {
        this.paginas = paginas;
    }

    public void setCantPagMP(int cantPagMP) {
        this.cantPagMP = cantPagMP;
    }

    public void setCantPagMS(int cantPagMS) {
        this.cantPagMS = cantPagMS;
    }

    public int getCantPagMP() {
        return cantPagMP;
    }

    public int getCantPagMS() {
        return cantPagMS;
```

```
    }

    public void addMP() {
        this.cantPagMP++;
        this.cantPagMS--;

        // Bloqueado
        if (!Controller.colaProcesos.contains(this)) {
            if (this.cantPagMP >= this.getMitad()) {
                this.setEstado("Bloqueado");
            } else {
                this.setEstado("Suspendido/Bloqueado");
            }
            // Listo
        } else {
            if (this.cantPagMP >= this.getMitad()) {
                this.setEstado("Listo");
            } else {
                this.setEstado("Suspendido/Listo");
            }
        }

        // Si todas sus paginas estan en MS y sumas en MP, pasa a la cola de MP
        if (!Controller.colaMemoriaPrincipal.contains(this)) {
            Controller.colaMemoriaPrincipal.offer(this);
        }

    }

    public int getMitad() {
```

```
        return ((int) Math.ceil((double) this.getPaginas().length / 2));
    }

    public void substractMP() {
        this.cantPagMP--;
        this.cantPagMS++;

        // Si las paginas en MP < mitad => suspendido
        if (this.isSuspendido()) {
            if (Controller.colaProcesos.contains(this)) {
                this.setEstado("Suspendido/Listo");
                Controller.actualizarMemorias();
            } else {
                this.setEstado("Suspendido/Bloqueado");
                Controller.actualizarMemorias();
            }
            // Si se quitó la ultima página de MP
        } else if (this.cantPagMP == 0) {
            Controller.colaMemoriaPrincipal.remove(this);
            if (Controller.colaProcesos.contains(this)) {
                this.setEstado("Suspendido/Listo");
                Controller.actualizarMemorias();
            } else {
                this.setEstado("Suspendido/Bloqueado");
                Controller.actualizarMemorias();
            }
        }
    }

    public void modificarPagina(boolean flag) {
```

```
        if (flag) {
            this.addMP();
        } else {
            this.substractMP();
        }
        Controller.actualizarMemorias();
    }

    public boolean isSuspendido(){
        return this.getCantPagMP() < this.getMitad();
    }

    public void eliminarPagina(Pagina pagina){
        for (int i = 0; i < Controller.memoriaSecundaria.length; i++) {
            if(pagina.equals(Controller.memoriaSecundaria[i].getPagina())){
                Controller.memoriaSecundaria[i].setPagina(null);
            }
        }
    }
}
```

CLASE PAGINA

```
package model;

import controller.Controller;

/**
 *
 * @author Edgar Telles
```

Catedra: Sistemas Operativos
Universidad: UMG
Repositorio de GITHUB

Alumno: 7691-19-11984 Victor Hugo Guerra Rodríguez
7691-15-9407 Edgar Adolfo Telles Veliz
https://github.com/v1ct0r-gu3rr4/sistemasOperativos.git

```
*/

public class Pagina {

    private int numPagina;

    private int tamaño;

    private boolean fragmentacion;

    private int tamañoFragmentacion;

    private boolean inMemoriaPrincipal;

    private int IDProceso;


    public Pagina(int numPagina, int tamaño, int IDProceso) {

        this.IDProceso = IDProceso;

        this.numPagina = numPagina;

        this.tamaño = tamaño;

        // Si hay fragmentacion

        if(tamaño < Controller.tamañoPagina) {

            this.fragmentacion = true;

            this.tamañoFragmentacion = Controller.tamañoPagina - this.tamaño;

        } else {

            this.fragmentacion = false;

            this.tamañoFragmentacion = 0;

        }

    }


    public int getNumPagina() {

        return numPagina;

    }


    public int getTamaño() {

        return tamaño;

    }
```

```
public boolean isFragmentacion() {
    return fragmentacion;
}

public int getTamañoFragmentacion() {
    return tamañoFragmentacion;
}

public boolean isInMemoriaPrincipal() {
    return inMemoriaPrincipal;
}

public int getIDProceso(){
    return IDProceso;
}

public void setNumPagina(int numPagina) {
    this.numPagina = numPagina;
}

public void setTamaño(int tamaño) {
    this.tamaño = tamaño;
}

public void setFragmentacion(boolean fragmentacion) {
    this.fragmentacion = fragmentacion;
}

public void setTamañoFragmentacion(int tamañoFragmentacion){
```

```
        this.tamañoFragmentacion = tamañoFragmentacion;
    }

    public void setInMemoriaPrincipal(boolean inMemoriaPrincipal) {
        Controller.modificarProceso(this.IDProceso, inMemoriaPrincipal);
        this.inMemoriaPrincipal = inMemoriaPrincipal;
        Controller.actualizarMemorias();
    }

    public void setIDProceso(int IDProceso) {
        this.IDProceso = IDProceso;
    }

    public void crearSetInMemoriaP(boolean inMemoriaP){
        this.inMemoriaPrincipal = inMemoriaP;
    }
}
```

CLASE MARCO

```
package model;

/**
 *
 * @author Edgar Telles
 */
public class Marco {
    private String dirFisica;
    private int numMarco;
    private Pagina pagina;
```

```
    private boolean inMemoriaPrincipal;

    public Marco(String dirFisica, int numMarco, Pagina pagina, boolean inMemoriaPrincipal) {
        this.dirFisica = dirFisica;
        this.numMarco = numMarco;
        this.pagina = pagina;
        this.inMemoriaPrincipal = inMemoriaPrincipal;
    }

    public String getDirFisica() {
        return dirFisica;
    }

    public int getNumMarco() {
        return numMarco;
    }

    public Pagina getPagina() {
        return pagina;
    }

    public boolean isInMemoriaPrincipal() {
        return inMemoriaPrincipal;
    }

    public void setDirFisica(String dirFisica) {
        this.dirFisica = dirFisica;
    }

    public void setNumMarco(int numMarco) {
```

```
        this.numMarco = numMarco;
    }

    public void setPagina(Pagina pagina) {
        this.pagina = pagina;
    }

    public void setInMemoriaPrincipal(boolean inMemoriaPrincipal) {
        this.inMemoriaPrincipal = inMemoriaPrincipal;
    }
}
```

CLASE FONQUINOS

```
package model;

import controller.Controller;

/**
 *
 * @author Edgar Telles
 */
public class FonQuinOS {

    public FonQuinOS() {
        Controller controlador = new Controller();
        controlador.initSO();
    }

    public static void main(String[] args) {
        FonQuinOS SO = new FonQuinOS();
```

Catedra: Sistemas Operativos
Universidad: UMG
Repositorio de GITHUB

Alumno: 7691-19-11984 Victor Hugo Guerra Rodríguez
7691-15-9407 Edgar Adolfo Telles Veliz
https://github.com/v1ct0r-gu3rr4/sistemasOperativos.git

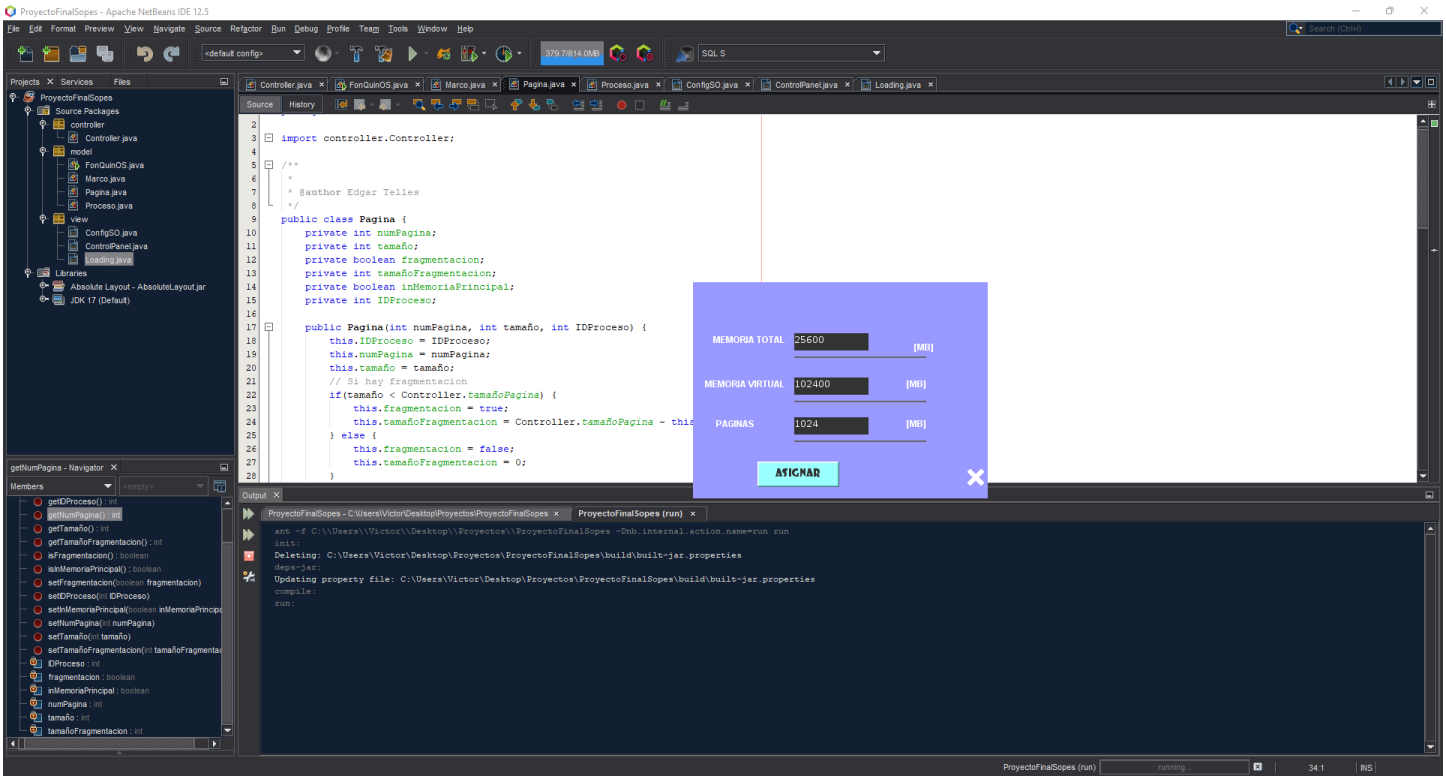
```
    }


}
```

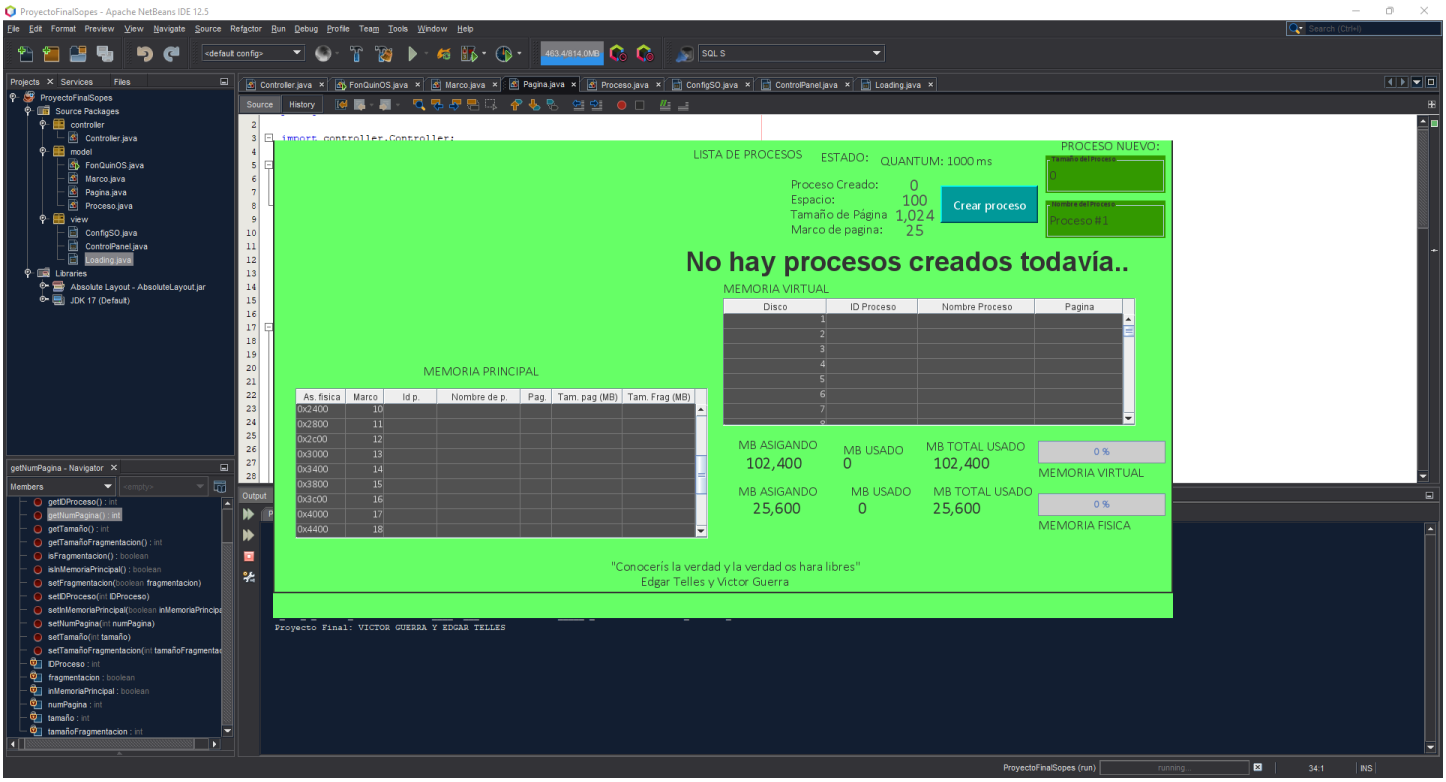
Catedra: Sistemas Operativos
Universidad: UMG
Repositorio de GITHUB

Alumno: 7691-19-11984 Victor Hugo Guerra Rodríguez
7691-15-9407 Edgar Adolfo Telles Veliz

https://github.com/v1ct0r-gu3rr4/sistemasOperativos.git

CORRIDA DEL PROGRAMA



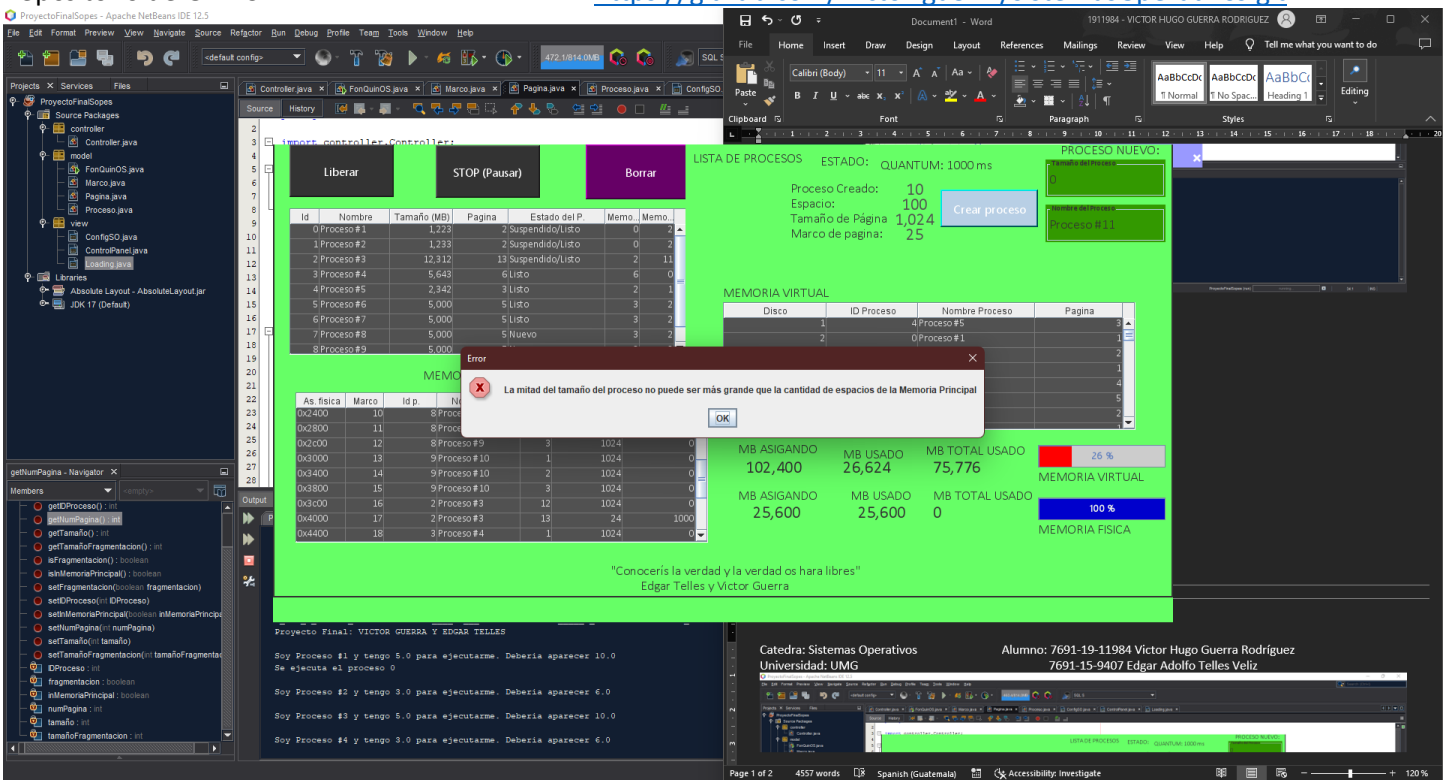


Cuando ingresamos valores muy grandes

Catedra: Sistemas Operativos
Universidad: UMG
Repositorio de GITHUB

Alumno: 7691-19-11984 Victor Hugo Guerra Rodríguez
7691-15-9407 Edgar Adolfo Telles Veliz

https://github.com/v1ct0r-gu3rr4/sistemasOperativos.git



Podemos observar que cuando se llena ambas barras se van coloreando y liberando posteriormente