Universidad Nacional Autónoma de Honduras
en el Valle de Sula

Facultad de Ingeniería

Departamento de Ingeniería en Sistemas

Informe proyecto final simulación de memoria

Asignatura: Sistemas operativos II 1800
Catedrático: Ing. Mario Pon

Alumno: José Daniel Sánchez Madrid
Cuenta: 20212020720

25 de abril 2024

Marco teórico

Particionamiento fijo de memoria: para este proyecto se permite al usuario especificar la cantidad de particiones en que se segmentará la memoria y el tamaño de cada partición, se llama particionamiento fijo ya que el tamaño especificado a cada partición no cambiará, una ventaja de este enfoque es que reduce la fragmentación externa.

Políticas

- Primer ajuste: Es la política mas sencilla, toma el proceso a ubicar y recorre la memoria en busca de una partición en la cual el proceso se pueda alojar (de mayor o igual tamaño que el proceso), sin importar si esta partición es demasiado grande para este proceso lo cual a veces mala utilización de particiones grandes de la memoria.

- Mejor ajuste: Tomando en cuenta la memoria que requiere un proceso, con esta política se analizan todas las particiones y se decide asignar el proceso en la partición más pequeña en la cual este quepa, en caso de haber dos o más particiones iguales, se usa la política de primer ajuste

Programa

Llenando información inicial.

1. Digitar una cantidad de memoria total.
2. Seleccionar la política a utilizar.
3. Digitar un intervalo de tiempo
4. Escribe la cantidad de particiones y presiona el botón “Crear”
5. Luego especifica el tamaño de cada partición, por defecto estas tienen el mismo tamaño.
6. Presiona “Siguiente”

Registrando procesos e iniciando simulación.

1. Llena los campos de memoria requerida y nombre del primer proceso, luego.

2. Presiona “Agregar Proceso” para un nuevo proceso o “Iniciar Simulación”.

Simbología:

- ParticiónID: Es una referencia hacia las particiones que cada proceso guarda será -1 cuando este proceso está en espera de memoria, de lo contrario tendrá el valor del número de la partición a la cual está asignado.
- Estado: será de valor 1 cuando el proceso está en ejecución, 2 cuando está en espera de memoria, 3 cuando ha terminado.

Errores:

- El programa mostrará errores con mensajes de advertencias significativas para el usuario como: La suma de las particiones no coincide con el total de memoria (especificando la diferencia).
- La cantidad máxima de particiones en cuanto a la memoria no debe ser mayor al 10% y no mayor a 10 particiones.
- No se admiten tamaños negativos de particiones
- Antes de agregar otra fila para otro proceso o para iniciar simulación se debe completar los datos de todas las filas que se hayan creado hasta el momento.
- El tamaño del proceso es demasiado grande y no podrá ser ejecutado.
- Ingrese solo números en la columna “Memoria requerida”.

Repositorio del proyecto

Para su registro en este repositorio se ha llevado el control del proyecto mediante un repositorio remoto de GitHub

https://github.com/DMadrid03/ProyectoSO2.git

Código

Clases de manejo lógico

```
internal class Particion
 {
     public int id { get; set; }
     public int size { get; set; }
     public bool ocupado { get; set; }

     public Particion(DataRow fila)
     {
         id = int.Parse(fila["Numero"].ToString());
         size = int.Parse(fila["tamaño"].ToString());
         ocupado = bool.Parse(fila["ocupado"].ToString());
     }
     public void liberarParticion()
     {
         this.ocupado = false;
     }


 }




internal class Proceso
{
    public int id { get; set; }
    public String Nombre { get; set; }
    public int Duracion { get; set; }
    public int MemoriaRequerida { get; set; }
    public int Estado { get; set; }
    public int TiempoTranscurrido { get; set; }
    public int TiempoInicio { get; set; }
    public int ParticionID { get; set; }

    public Proceso(DataRow fila)
    {
        id = int.Parse(fila["id"].ToString());
        Nombre = fila["Nombre"].ToString();
        Duracion = int.Parse(fila["Duración"].ToString());
        MemoriaRequerida = int.Parse(fila["memoria Requerida"].ToString());
        Estado =int.Parse(fila["estado"].ToString());//al inicio en estado 2 (En espera de
memori)
        TiempoTranscurrido = int.Parse(fila["tiempo transcurrido"].ToString());
        TiempoInicio = int.Parse(fila["Tiempo Inicio"].ToString());
        ParticionID = int.Parse(fila["particionID"].ToString());
    }

}
```

```
internal class Gestor
{
    public Particion particion { get; set; }
    public Proceso proceso { get; set; }
    public List<Proceso> ProcesosList;
    public List<Particion> ParticionesList;
    private DataTable tabProcesos, tabParticiones;
    public int procesosTerminados;

    public Gestor(DataTable tabProcesos,DataTable tabParticiones)
    {
        procesosTerminados = 0;
       this.tabProcesos = tabProcesos;
       this.tabParticiones = tabParticiones;
        ProcesosList = new List<Proceso>();
        ParticionesList = new List<Particion>();

    }
    public List<Proceso> getProcesosList()
    {
        ProcesosList.Clear();
        for(int i =0; i<tabProcesos.Rows.Count;i++)
        {
            Proceso proceso = new Proceso(tabProcesos.Rows[i]);
            ProcesosList.Add(proceso);
        }
        return ProcesosList;
    }
    public List<Particion> getParticionesList()
    {
        ParticionesList.Clear();
        //crear lista de particiones
        for (int i = 0; i < tabParticiones.Rows.Count; i++)
        {
            Particion particion = new Particion(tabParticiones.Rows[i]);
            ParticionesList.Add(particion);
        }
        return ParticionesList;
    }
    public DataTable getTableProcesos()
    {
        if(tabProcesos.Rows.Count >0)tabProcesos.Rows.Clear();
        DataRow fila;
        foreach (Proceso p in ProcesosList)
        {
            fila = tabProcesos.NewRow();
            fila["ID"] = p.id;
            fila["nombre"] = p.Nombre;
            fila["Duración"] = p.Duracion;
            fila["Memoria Requerida"] = p.MemoriaRequerida;
            fila["Estado"] = p.Estado;
            fila["Tiempo Transcurrido"] = p.TiempoTranscurrido;
            fila["particionID"] = p.ParticionID;
            fila["Tiempo Inicio"] = p.TiempoInicio;
            tabProcesos.Rows.Add(fila);
        }
        return tabProcesos;
    }
    public DataTable getTableParticiones()
    {
        tabParticiones.Rows.Clear();
        DataRow fila;
```

```
        foreach(Particion p in ParticionesList)
        {
            fila = tabParticiones.NewRow();
            fila["Numero"] = p.id;
            fila["Tamaño"] = p.size;
            fila["Ocupado"] = p.ocupado;
            tabParticiones.Rows.Add(fila);
        }

        return tabParticiones;
    }
    public bool verificarProcesoSaliente()
    {
        bool SalieronProcesos = false;
        int pt = 0;
        for(int i=0; i < ProcesosList.Count; i++)
        {
            proceso = ProcesosList[i];
            if(proceso.Estado ==1 && proceso.Duracion ==proceso.TiempoTranscurrido -
proceso.TiempoInicio)
            {//proceso en ejecución, y ya paso su tiempo de ejecución
                ParticionesList[proceso.ParticionID - 1].liberarParticion();
                proceso.Estado = 3;
                SalieronProcesos = true;
                proceso.ParticionID = -1;//sacar proceso
                ProcesosList[i] = proceso;
            }
            if(proceso.Estado != 3 && particionesLibres()!=ParticionesList.Count)
            {//el proceso está en memoria y se seguirá ejecutando
                proceso.TiempoTranscurrido++;
            }
            if (proceso.ParticionID == -1)
                pt++;//conteo de procesos terminados
        }
        procesosTerminados = pt;
        return SalieronProcesos;
    }
    public int particionesLibres()
    {
        int pl = 0;
        foreach(Particion p in ParticionesList)
        {
            if (p.ocupado == false)
                pl++;
        }
        return pl;
    }
    public void BestFit()
    {//va a asignar el/los procesos libres en las particiones de memoria que estén libres usando
mejor ajuste
        foreach(Proceso proc in ProcesosList)
        {
            if(proc.Estado ==2){
                proceso = proc;
                int diferencia = 100000000;
                int particionIndex = -1;
                for (int i=0; i<ParticionesList.Count;i++)
                {
                    particion = ParticionesList[i];
                    if ((!particion.ocupado) && particion.size >= proceso.MemoriaRequerida)
                    {
                        if (particion.size - proceso.MemoriaRequerida < diferencia)
```

```
                        {//definir la partición mas óptima para ese proceso
                            diferencia = particion.size - proceso.MemoriaRequerida;
                            proceso.ParticionID = particion.id;
                            particionIndex = i;
                        }
                        proceso.Estado = 1;
                    }
                }
                if(particionIndex != -1)
                {//ese proceso encontró una particion donde ubicarse
                    proc.ParticionID = proceso.ParticionID;//asignar el proceso a esa partición
                    proc.TiempoInicio = proceso.TiempoTranscurrido - 1;//guardar el segundo en
que ese proceso entró a memoria
                    ParticionesList[particionIndex].ocupado = true; //marcar la partición como
ocupada
                }

            }
        }
    }
    public void firstFit()
    {
        foreach(Proceso pro in ProcesosList)
        {//asignar los procesos libres a los huecos libres de memoria usando primer ajuste
            if(pro.Estado == 2)
            {
                foreach(Particion par in ParticionesList)
                {
                    if(par.ocupado == false && pro.MemoriaRequerida <= par.size)
                    {
                        pro.ParticionID = par.id;//asignar directamente el proceso a la partición
                        pro.TiempoInicio = pro.TiempoTranscurrido;
                        pro.Estado = 1;//proceso en estado (ejecutando)
                        par.ocupado = true;
                        break;//dejar de buscar particiones para ese proceso
                    }
                }
            }
        }
    }
    public Particion particionMasGrande()
    {

        int size=0,index=0;
        for (int i = 0; i < ParticionesList.Count ; i++)
        {
            if (ParticionesList[i].size > size)
            {
                index = i;
                size = ParticionesList[i].size;
            }
        }
        return ParticionesList[index];
    }
}
```

Clase formulario de simulación

```
public partial class frmSimulacion : Form
{
    DataRow filaInfo;
    DataTable tabParticiones;
    DataTable tabProcesos;
    int particiones;
    int min;
    int max;
    Random random;
    List<Particion> particionesList;
    List<Proceso> procesosList;
    Gestor gestor;
    bool simulacion;
    Thread hiloPrincipal;
    public frmSimulacion(DataRow fila, DataTable tbPart)
    {//recibe un arreglo lineal con la información inicial que ingresó el usuario y recibe la tabla de particiones
        InitializeComponent();
        simulacion = false;
        tabParticiones = tbPart;
        filaInfo = fila;
        tabProcesos = new DataTable();
        gestor = new Gestor(tabProcesos, tabParticiones);



        particiones = int.Parse(filaInfo["nParticiones"].ToString()) + 1;
        min = int.Parse(filaInfo["min"].ToString());
        max = int.Parse(filaInfo["max"].ToString());
        random = new Random();


        panMemoria.BackColor = Color.Gray;


    }
```

```
    private void frmSimulacion_Load(object sender, EventArgs e)
    {
        //configurar el datagridView y la tabla para guardar procesos
        tabProcesos.Columns.Add("ID");
        tabProcesos.Columns.Add("Nombre");
        tabProcesos.Columns.Add("Duración");
        tabProcesos.Columns.Add("Memoria Requerida");
        tabProcesos.Columns.Add("Estado");
        tabProcesos.Columns.Add("Tiempo Transcurrido");
        tabProcesos.Columns.Add("particionID");
        tabProcesos.Columns.Add("Tiempo Inicio");
        dgvProcesos.DataSource = tabProcesos;

        dgvProcesos.Columns["ID"].ReadOnly = true; dgvProcesos.Columns["ID"].Width = 35;
        dgvProcesos.Columns["Duración"].ReadOnly = true; dgvProcesos.Columns["Memoria
Requerida"].Width = 70;
        dgvProcesos.Columns["Estado"].ReadOnly = true; dgvProcesos.Columns["Estado"].Width = 50;
        dgvProcesos.Columns["Tiempo Transcurrido"].ReadOnly = true; dgvProcesos.Columns["Tiempo
Transcurrido"].Width = 90;
        dgvProcesos.Columns["Nombre"].Width = 250; dgvProcesos.Columns["Duración"].Width = 70;
        dgvProcesos.Columns["Tiempo Inicio"].Visible = false;
        dgvProcesos.Columns["ParticionID"].ReadOnly = true;

        //agregar fila para el primer proceso
        DataRow fila = tabProcesos.NewRow();
        fila["ID"] = 1;
        fila["Duración"] = random.Next(min, max);
        fila["Estado"] = 2;
        fila["Tiempo Transcurrido"] = 0;
        fila["tiempo inicio"] = -1;
        fila["ParticionID"] = 0;
        tabProcesos.Rows.Add(fila);
        dibujarParticiones();
    }
    private void dibujarParticiones()
    {
```

```
        panMemoria.Controls.Clear();

        // Crear y agregar los paneles internos
        for (int i = 0; i < particiones-1; i++)
        {
            Panel panelParticion = new Panel();
            panelParticion.BackColor = System.Drawing.Color.Green;
            panelParticion.Dock = DockStyle.Top;
            panelParticion.Height = panMemoria.Height / particiones;
            panelParticion.BorderStyle = BorderStyle.FixedSingle;
            panelParticion.Name = "particion" + i;



            // Crear y configurar el label dentro del panel interno
            Label lblTexto = new Label();
            lblTexto.Font = new Font("Verdana", 11, FontStyle.Regular);
            lblTexto.Text = panelParticion.Name + "      porcentaje usado:0%           tamaño: "
+ tabParticiones.Rows[i ]["tamaño"];
            lblTexto.AutoSize = true;
            lblTexto.Location = new Point(panelParticion.Width / 5, panelParticion.Height / 4);

            // Agregar el label al panel interno
            panelParticion.Controls.Add(lblTexto);

            // Agregar el panel interno al contenedor
            panMemoria.Controls.Add(panelParticion);
        }
    }
    private void actualizarGestor()
    {//actualizar las listas de procesos y particiones con las que trabaja el gestor
        //en base a las versiones actuales (de las tablas)
        gestor = new Gestor(tabProcesos, tabParticiones);
        gestor.getParticionesList();
        gestor.getProcesosList();
    }
```

```
    private void btnAgregar_Click(object sender, EventArgs e)
    {
        DataRow fila;

        if (validarProceso())
        {
            fila = tabProcesos.NewRow();
            fila["ID"] = int.Parse(tabProcesos.Rows[tabProcesos.Rows.Count - 1]["ID"].ToString())
+ 1; //calcular el id
            fila["Duración"] = random.Next(min, max); //generar automaticamente
            fila["Estado"] = 2;
            fila["Tiempo Transcurrido"] = 0;
            fila["particionID"] = 0;
            fila["tiempo inicio"] = -1;
            tabProcesos.Rows.Add(fila);
        }

    }



    private void dgvProcesos_CellValidating(object sender, DataGridViewCellValidatingEventArgs e)
    {//validaciones de rango y racionalidad para el campo memoria requerida
        if (e.ColumnIndex == dgvProcesos.Columns["Memoria Requerida"].Index)
        {
            string valor = e.FormattedValue.ToString();

            // Intenta convertir el valor a un número
            int numero;
            if (!int.TryParse(valor, out numero))
            {
                // Si no se puede convertir a número, muestra un mensaje de error
                MessageBox.Show("Ingrese solo números en la columna 'Memoria Requerida'.", "Error
de validación",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
                dgvProcesos.CancelEdit();
            }
            if (numero < 0)
            {
                MessageBox.Show("No permitido valores negativos en Memoria requerida", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                dgvProcesos.CancelEdit();
            }
        }
    }

    private            void            dgvProcesos_EditingControlShowing(object            sender,
DataGridViewEditingControlShowingEventArgs e)
    {
    }
    private void dgvProcesos_KeyPress(object sender, KeyPressEventArgs e)
    {
    }
    private void FlowLayoutPanelMemoria_Resize(object sender, EventArgs e)
    {
    }
    private void flowLayoutPanelMemoria_ControlAdded(object sender, ControlEventArgs e)
    {
    }

    private int procesosNoVacios()
    {
        //verificar que los datos que el usuario debe ingresar no estén vacíos

        foreach (DataRow row in tabProcesos.Rows)
        {
            if (row["Nombre"].ToString().Length == 0 || row["Memoria requerida"].ToString().Length
== 0)
                return -1;//significa que faltan datos

            if (int.Parse(row["Memoria requerida"].ToString()) == 0)
```

```
                return -2; //significa que se esta poniendo como 0 la memoria requerida
            //retorna -2 como valor bandera de que hay que envíar advertencia al usuario
        }
        return 0;
    }
    private bool validarProceso()
    {
        DataRow f = tabProcesos.NewRow();
        f = tabProcesos.Rows[tabProcesos.Rows.Count - 1];



        if (f["Nombre"].ToString().Length == 0 || f["Memoria Requerida"].ToString().Length == 0)
        {//verificar que al proceso anterior no le falten datos y que el tamaño no sea mayor que la partición más grande
            MessageBox.Show("Complete los datos del último proceso antes de agregar otro", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return false;
        }
        actualizarGestor();//reinicializar el gestor con los procesos agregados
        //hasta el momento (para que el procesosList esté actualizado)
        Particion p = gestor.particionMasGrande();
        if (int.Parse(f["Memoria Requerida"].ToString()) > p.size)
        {
            MessageBox.Show("El proceso requiere demasiada memoria y no podrá ser ejecutado", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            dgvProcesos.CurrentRow.Cells[3].Value = p.size;
            MessageBox.Show("Tamaño corregido al máximo permitido", "Información", MessageBoxButtons.OK, MessageBoxIcon.Information);
            return false;
        }
        return true;
    }
    private void btnIniciar_Click(object sender, EventArgs e)
    {
        int bandera = procesosNoVacios();
        if (bandera == -1)
```

```
        {
            MessageBox.Show("Faltan datos de los procesos", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }
        if (bandera == -2)
        {
            MessageBox.Show("La cantidad de memoria requerida de un proceso no puede ser cero",
"Error con la memoria requerida", MessageBoxButtons.OK, MessageBoxIcon.Stop);
            return;
        }
        else
        {
            if (validarProceso())
                simulacion = true;
            else
                return;
            btnAgregar.Visible = false;
            btnIniciar.Visible = false;
            dgvProcesos.ReadOnly = true;
            btnDetener.Visible = true;
            //una vez todos los datos son correctos iniciar el hilo
            hiloPrincipal = new Thread(new ThreadStart(accionesPeriodicas));
            //el método accionesPeriodicas() tiene las llamadas a otros métodos que componen las
funciones
            //que se deben de ejecutar cada segundo
            hiloPrincipal.Start();
        }



    }
    public void accionesPeriodicas() //método vinculado al hilo
    {
        while (simulacion)
        {
            actualizarGestor();
```

```
                if (gestor.verificarProcesoSaliente() || gestor.particionesLibres() > 0)
                {
                    if (filaInfo["Politica"].Equals("Primer Ajuste"))
                    {
                        gestor.firstFit();
                    }
                    else
                    {
                        gestor.BestFit();
                    }
                }

            //actualizar las listas del formulario en base a las del objeto Gestor
            //que acaba de asignar o sacar procesos de la memoria.
            particionesList = gestor.ParticionesList;
            procesosList = gestor.ProcesosList;

            Thread.Sleep(1000); //suspender el hilo por un segundo

            dgvProcesos.Invoke((MethodInvoker)delegate
            {//llamada asíncrona al manejador de los objetos de la interfaz gráfica para poder
modificarlos
                dgvProcesos.DataSource = null;
                tabProcesos.Rows.Clear();
                tabProcesos = gestor.getTableProcesos();
                dgvProcesos.DataSource = tabProcesos;
                dgvProcesos.Refresh();//refrescar el DataGridView para que se reflejen los cambios
de
                //tiempo transcurrido ,particion que ocupa y estado.

            });

            tabParticiones.Rows.Clear();
            tabParticiones = gestor.getTableParticiones();
```

```
            //recorrer los procesos y llamar a la función de actualizar paneles para la particionid
que ocupan
            foreach(Proceso proceso in procesosList)
            {
                if(proceso.ParticionID != -1)//proceso en ejecución.
                {
                    if(proceso.ParticionID >0 )
                    {
                        actualizarPanel(proceso.ParticionID - 1, proceso);
                    }
                }
            }

            panelesLibres();



            if (gestor.procesosTerminados == procesosList.Count) //verificar que ya no hayan
procesos para ejecutar
                simulacion = false;


        }
    }
    private void actualizarPanel(int particionID, Proceso proceso)
    {//pone la información del proceso en su panel correlativo que representa su partición.
        float usado = (proceso.MemoriaRequerida / particionesList[particionID].size) * 100;
        panMemoria.Invoke((MethodInvoker)delegate
        {
            panMemoria.Controls[particionID].BackColor = Color.Yellow;
            panMemoria.Controls[particionID].Controls[0].Text = "Particion " + particionID + "
Aloja proceso: " +
            proceso.Nombre +
            "       Usado: " + usado +
            "%             tamaño: " + particionesList[particionID].size + "       Tiempo: " +
proceso.TiempoTranscurrido;
        });
```

```
    }
    private void panelesLibres()
    {
        //pintar paneles libres
        for (int i =0;i<particionesList.Count;i++)
        {
            if (!particionesList[i].ocupado)
            {
                panMemoria.Invoke((MethodInvoker)delegate
                {
                    panMemoria.Controls[i].BackColor = Color.Green;
                    panMemoria.Controls[i].Controls[0].Text = "Particion " + i + " Libre";
                });
            }
        }
    }
    private void btnDetener_Click(object sender, EventArgs e)
    {
        particionesList = gestor.ParticionesList;
        procesosList = gestor.ProcesosList;
        simulacion = false;
    }
}
```

Clase formulario de información inicial

```
using System.Data;
namespace ProyectoSO2
{
    public partial class frmInfo : Form
```

```
{
    DataTable tabParticiones;
    int tamanoTotal;

    public frmInfo()
    {
        InitializeComponent();
        tabParticiones = new DataTable();
        tabParticiones.Columns.Add("Numero"); tabParticiones.Columns.Add("Tamaño");
        tabParticiones.Columns.Add("Ocupado");
        dgvParticiones.DataSource = tabParticiones;
    }
    private void LimitarATextoNumerico(TextBox textBox, KeyPressEventArgs e)
    {
        // Verificar si la tecla presionada es un número o una tecla de control
        if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar))
        {
            e.Handled = true; // Ignorar la entrada de caracteres no numéricos
        }
    }

    private void txtTamanoTotal_KeyPress(object sender, KeyPressEventArgs e)
    {
        LimitarATextoNumerico(txtTamanoTotal, e);
    }
    private void txtNParticiones_keyPress(object sender, KeyPressEventArgs e)
    {
        LimitarATextoNumerico(txtNParticiones, e);
    }
    private void txtMin_KeyPress(object sender, KeyPressEventArgs e)
    {
        LimitarATextoNumerico(txtMin, e);
    }
    private void txtMax_KeyPress(object sender, KeyPressEventArgs e)
    {
        LimitarATextoNumerico(txtMax, e);
    }

    private void frmInfo_Load(object sender, EventArgs e)
    {
        cmbPolitica.SelectedIndex= 0;
        txtNParticiones.Text = "2";
        btnSiguiente.Visible = false;
        btnValidar.Visible = false;
        dgvParticiones.Columns["Numero"].ReadOnly = true;
        dgvParticiones.Columns["Tamaño"].ReadOnly = true;
        dgvParticiones.Columns["ocupado"].Visible = false;
        dgvParticiones.AllowUserToAddRows = false;
        dgvParticiones.AllowUserToDeleteRows = false;

    }

    private void txtNParticiones_TextChanged(object sender, EventArgs e)
    {

    }

    private void txtNParticiones_KeyUp(object sender, KeyEventArgs e)
    {
    }

    private void btnCrear_Click(object sender, EventArgs e)
```

```
        {
            if (int.Parse(txtNParticiones.Text) > 10)
            {
                MessageBox.Show("Muchas particiones para mostrar, el máximo son 10", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                txtNParticiones.Text = "10";
                return;
            }
            if (int.Parse(txtNParticiones.Text) > int.Parse(txtTamanoTotal.Text) / 10)
            {
                MessageBox.Show("Demasiadas particiones, el tamaño máximo es: " +
(int.Parse(txtTamanoTotal.Text)/10) + " (10% del tamaño total) ", "Error, pruebe aumentar el
tamaño total de memoria", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (txtTamanoTotal.Text.Length == 0 || txtTamanoTotal.Text == " " ||
int.Parse(txtNParticiones.Text)==0)
            {
                MessageBox.Show("Error, ingrese antes el tamaño total de la memoria y la cantidad
de particiones mayor a cero");
                return;
            }
            else
            {
                try
                {
                    tamanoTotal = int.Parse(txtTamanoTotal.Text);
                    int n = int.Parse(txtNParticiones.Text);
                    for (int i = 1; i <= n; i++)
                    {
                        tabParticiones.Rows.Add(i, tamanoTotal/n,false);
                    }
                }
                catch (Exception)
                {
                    MessageBox.Show("error ");
                }

                btnCrear.Enabled = false;
                txtTamanoTotal.Enabled = false;
                dgvParticiones.Columns["Tamaño"].ReadOnly = false;
                btnValidar.Visible = true;
            }
        }

        private void dgvParticiones_CellContentClick(object sender, DataGridViewCellEventArgs e)
        {

        }

        private void btnSiguiente_Click(object sender, EventArgs e)
        {
            DataTable dt = new DataTable();
            dt.Columns.Add("tamTot", typeof(float));
            dt.Columns.Add("politica", typeof(String));
            dt.Columns.Add("nParticiones", typeof(int));
            dt.Columns.Add("min", typeof(int));
            dt.Columns.Add("max", typeof(int));

            DataRow fila = dt.NewRow();
            fila["tamTot"] = tamanoTotal;
            fila["politica"] = cmbPolitica.SelectedItem.ToString();
```

```
            fila["nParticiones"] = int.Parse(txtNParticiones.Text);
            fila["min"] = int.Parse(txtMin.Text);
            fila["max"] = int.Parse(txtMax.Text);

            frmSimulacion frmSim = new frmSimulacion(fila,tabParticiones);
            frmSim.Show();
        }

        private void dgvParticiones_Enter(object sender, EventArgs e)
        {

        }

        private void dgvParticiones_CellValidating(object sender,
DataGridViewCellValidatingEventArgs e)
        {
            if (e.ColumnIndex == dgvParticiones.Columns["Tamaño"].Index)
            {
                // Obtén el valor de la celda que se está validando
                string valor = e.FormattedValue.ToString();

                // Intenta convertir el valor a un número
                int numero;
                if (!int.TryParse(valor, out numero))
                {
                    // Si no se puede convertir a número, muestra un mensaje de error
                    MessageBox.Show("Ingrese solo números en la columna 'Tamaño '.", "Error de
validación",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);

                    // Cancela la edición de la celda
                    dgvParticiones.CancelEdit();
                }
                if (numero < 0)
                {
                    DialogResult result = MessageBox.Show("El tamaño de la partición no puede ser
negativo. ¿Desea cambiarlo a positivo?", "Error de datos de entrada", MessageBoxButtons.OKCancel,
MessageBoxIcon.Error);
                    if (result == DialogResult.OK)
                    {
                        tabParticiones.Rows[e.RowIndex][e.ColumnIndex] = Math.Abs(numero);
                        dgvParticiones.Refresh();
                    }
                    else
                    {
                        dgvParticiones.CancelEdit();
                    }
                }

            }
        }


        private void dgvParticiones_CellValueChanged(object sender, DataGridViewCellEventArgs e)
        {


        }

        private void btnValidar_Click(object sender, EventArgs e)
        {
```

```
            if (cmbPolitica.SelectedIndex != -1 && txtMax.Text.Length != 0 && txtMin.Text.Length
!=0)
            {
                int totalingresado = 0;
                //validar que la suma de las particiones sea = total memoria
                for (int i = 0; i < int.Parse(txtNParticiones.Text); i++)
                {
                    int n = int.Parse(tabParticiones.Rows[i]["Tamaño"].ToString());
                    if (n < 0)
                        n = n * -1;
                    totalingresado += +n;
                }

                if (totalingresado > tamanoTotal)
                {
                    MessageBox.Show("La suma del tamaño de las particiones excede el total de
memoria por: " + (totalingresado - tamanoTotal), "Error", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
                    return;
                }
                if (totalingresado < tamanoTotal)
                {
                    MessageBox.Show("Debe agregar " + (tamanoTotal - totalingresado) + " espacio
en las particiones", "Falta espacio para completar el total de memoria");
                    return;
                }
                if(int.Parse(txtMax.Text)<= int.Parse(txtMin.Text))
                {
                    MessageBox.Show("Error en el intervalo","Error",MessageBoxButtons.OK,
MessageBoxIcon.Stop);
                    return;
                }
                //btnSiguiente_Click(sender, new EventArgs());
                DataTable dt = new DataTable();
                dt.Columns.Add("tamTot", typeof(float));
                dt.Columns.Add("politica", typeof(String));
                dt.Columns.Add("nParticiones", typeof(int));
                dt.Columns.Add("min", typeof(int));
                dt.Columns.Add("max", typeof(int));

                DataRow fila = dt.NewRow();
                fila["tamTot"] = tamanoTotal;
                fila["politica"] = cmbPolitica.SelectedItem.ToString();
                fila["nParticiones"] = int.Parse(txtNParticiones.Text);
                fila["min"] = int.Parse(txtMin.Text);
                fila["max"] = int.Parse(txtMax.Text);

                frmSimulacion frmSim = new frmSimulacion(fila, tabParticiones);
                dgvParticiones.DataSource = null;
                frmSim.Show();

            }
            else
            {
                MessageBox.Show("Faltan Datos","Error",MessageBoxButtons.OK,MessageBoxIcon.Stop);
            }

        }
    }
}
```

Conclusiones

- Tras la implementación y análisis de las políticas de ubicación (primer ajuste y mejor ajuste), se observa que la elección de la política afecta significativamente el rendimiento del sistema.

- La implementación de una cola de procesos "en espera de memoria" junto con la política de primer en entrar, primer en ser servido (PEPS) permite una gestión dinámica de la asignación de memoria. Esta funcionalidad garantiza que los procesos no sean bloqueados indefinidamente, incluso si no pueden asignarse inmediatamente a una partición de memoria, lo que mejora la eficiencia y la fluidez del sistema.

- La política de mejor ajuste muestra una mejor utilización de la memoria al reducir la fragmentación interna, lo que se traduce en una mayor capacidad para alojar procesos en la memoria disponible.

- La representación gráfica de la memoria y los procesos proporciona una experiencia de usuario intuitiva y facilita la comprensión del estado del sistema en tiempo real.

Bibliografía

- Deitel, P. J., & Deitel, H. M. (2005). Cómo Programar en C# (2da ed.). Pearson Educación.

- Stallings, W. (2005). Sistemas operativos: aspectos internos y principios de diseño (7ma ed.). Pearson Educación.