Universidad Nacional Autónoma de Honduras

Facultad de Ingeniería

UNAH-Campus Comayagua

Sistemas Operativos 1

IS-412

Trabajo:

Informe Técnico Proyecto Gestor de Memoria Virtual Simplificado

Presentado a:

Ing. Elmer Padilla

Integrantes:

Erik Francisco Guillén Reyes

Número de cuenta:

20211920287

Sección:

1000

Comayagua, Comayagua

22 de abril de 2025

Introducción

Este proyecto implementa y simula tres algoritmos de sustitución de páginas en memoria: FIFO (First In, First Out) , LRU (Least Recent Used) y Óptimo . El proyecto está desarrollado en Python con una interfaz gráfica de usuario (GUI) utilizando Tkinter, lo que facilita la interacción con el usuario. La simulación permite ingresar una secuencia de referencias de páginas y un tamaño de marcos en la memoria, ejecutando los algoritmos de sustitución de páginas y mostrando el número de fallos de página.

Objetivos

Objetivo General:

Simular la gestión de memoria virtual en un sistema operativo, implementando y comparando algoritmos de reemplazo de páginas para analizar su eficiencia y comportamientos.

Objetivos Específicos:

- Implementar la segmentación de memoria en páginas y marcos, replicando el funcionamiento básico de la memoria virtual en los sistemas operativos modernos.
- Desarrollar simulaciones de los algoritmos FIFO y LRU, permitiendo observar cómo gestionan el reemplazo de páginas y cómo responden ante distintas cadenas de referencias.
- Incluir el algoritmo óptimo como referencia teórica, para comparar su rendimiento ideal frente a los algoritmos implementables en la práctica.

Algoritmos de Reemplazo de Páginas

1. Algoritmo FIFO (primero en entrar, primero en salir)

El algoritmo FIFO sigue el principio de reemplazar la página que ha estado en la memoria durante más tiempo. El proceso funciona de la siguiente manera:

Se mantiene una lista de las páginas en la memoria.

Cuando una nueva página necesita ser cargada y la memoria está llena, el algoritmo elimina la página que ingresó primero (es decir, la página más antigua).

El FIFO es sencillo y fácil de implementar, pero tiene el inconveniente de que no tiene en cuenta la frecuencia de uso de las páginas, lo que puede generar muchos fallos de página si las páginas más antiguas no han sido utilizadas recientemente.

2. Algoritmo LRU (menos usado recientemente)

El algoritmo LRU reemplaza la página que no ha sido utilizada durante más tiempo. Este algoritmo mantiene un registro de cuándo se accedió a cada página y elimina la página que no se ha usado durante el período más largo.

Cada vez que una página es referenciada, su "edad" se restablece. Cuando una nueva página necesita ser cargada y la memoria está llena, el algoritmo elimina la página con la edad más alta (la que no se ha usado recientemente).

El LRU tiene el beneficio de priorizar las páginas que se usan con más frecuencia, pero puede ser más costoso en términos de rendimiento si se debe realizar un seguimiento constante del tiempo de uso de las páginas.

3. Algoritmo Óptimo (Óptimo)

El algoritmo Óptimo es el algoritmo teóricamente más eficiente en términos de fallos de página, pero no es práctico de implementar en un entorno real, ya que requiere conocer las futuras referencias de páginas. El proceso funciona de la siguiente manera:

Se reemplaza la página que no se usará durante el mayor tiempo en el futuro.

Este algoritmo no genera errores de página innecesarios, ya que siempre toma la decisión de minimizar el número de errores futuros.

A pesar de ser ideal en términos de eficiencia, su implementación es solo de interés para fines comparativos, ya que en un sistema real no es posible predecir el futuro con precisión.

Diseño de la solución

Clase `Proceso`: Representa un proceso en memoria, con atributos para el valor (identificador de la página) y la edad.

```
9   class Proceso:
10      def __init__(self, valor):
11          self.valor = valor
12          self.edad = 0
13
14      def get_valor(self):
15          return self.valor
16
17      def set_edad(self, edad):
18          self.edad = edad
19
20      def get_edad(self):
21          return self.edad
```

Clase algoritmo FIFO (First In, First Out)

En este contexto se utiliza para reemplazar la página que lleva más tiempo en la memoria, eliminando la que fue cargada primero-

```python
#Clase para el algoritmo FIFO (First In First Out)
class AlgoritmoFIFO:
    def __init__(self, marcos):
        self.marcos = marcos

    def ejecutar(self, paginas):
        marcos = ["-"] * self.marcos
        puntero = 0
        fallos = []
        estado = []

        for pagina in paginas:
            if pagina not in marcos:
                marcos[puntero] = pagina
                puntero = (puntero + 1) % self.marcos
                fallos.append("F")
            else:
                fallos.append("/")
            estado.append(marcos[:])
        return estado, fallos
```

Clase algoritmo LRU (Least Recently Used)

Este reemplaza la página que no ha sido utilizada durante el mayor tiempo basándose en el historial de acceso a las páginas, tomando en cuenta que las paginas que accedidas recientemente tienen más probabilidad de ser usadas nuevamente en el futuro cercano.

```python
#Clase para el algoritmo LRU (Least Recently Used)
class AlgoritmoLRU:
    def __init__(self, marcos):
        self.marcos = marcos

    def ejecutar(self, paginas):
        marcos = []
        fallos = []
        estado = []

        for pagina in paginas:
            if pagina not in marcos:
                if len(marcos) < self.marcos:
                    marcos.append(pagina)
                else:
                    marcos.pop(0)
                    marcos.append(pagina)
                fallos.append("F")
            else:
                marcos.remove(pagina)
                marcos.append(pagina)
```

Clase algoritmo Optimo

Implementación del algoritmo **Optimo** el cual reemplaza la página que no se utilizará durante más tiempo en el futuro. Utilizado para comparar el rendimiento de los algoritmos de reemplazo, como seria en este caso FIFO y LRU, proporciona una referencia teórica ideal para los fallos de página.

```python
#Clase para el algoritmo Optimo
class AlgoritmoOptimo:
    def __init__(self, marcos):
        self.marcos = marcos

    def ejecutar(self, paginas):
        marcos = []
        fallos = []
        estado = []

        for i in range(len(paginas)):
            pagina = paginas[i]
            if pagina not in marcos:
                if len(marcos) < self.marcos:
                    marcos.append(pagina)
                else:
                    futuro = paginas[i+1:]
                    indices = [futuro.index(p) if p in futuro else float("inf") for p in marcos]
                    reemplazo = indices.index(max(indices))
                    marcos[reemplazo] = pagina
                fallos.append("F")
            else:
                fallos.append("/")
            estado.append(marcos[:] + ["-"] * (self.marcos - len(marcos)))
        return estado, fallos
```

```python
#Función para ejecutar el algoritmo
def ejecutar_algoritmos():
    entrada = entrada_paginas.get()
    try:
        paginas = [int(p.strip()) for p in entrada.split(",")]
    except:
        messagebox.showerror( title: "Error",  message: "Entrada inválida")
        return
    try:
        marcos = int(entry_marcos.get())
        if marcos <= 0:
            raise ValueError
    except:
        messagebox.showerror( title: "Error",  message: "Número de marcos inválido")
        return
    resultados = ""
    for nombre, algoritmo in [("FIFO", AlgoritmoFIFO(marcos)), ("LRU", AlgoritmoLRU(marcos)), ("Optimo", AlgoritmoOptimo(marcos))]:
        estado, fallos = algoritmo.ejecutar(paginas)
        transpuesta = np.array(estado).T.tolist()
        resultados += f"\n\nAlgoritmo: {nombre}\n"
        resultados += "  " + "  ".join(map(str, paginas)) + "\n"
        resultados += tabulate(transpuesta, tablefmt="fancy_grid") + "\n"
        resultados += "  " + "  ".join(fallos) + "\n"
        total_fallos = fallos.count("F")
        eficiencia = ((len(paginas) - total_fallos) / len(paginas)) * 100
        resultados += f"Fallos de página: {total_fallos}\n"
        resultados += f"Eficiencia: {eficiencia:.2f}%\n"
```

Interfaz Gráfica (GUI)

Utilice Tkinter para crear una interfaz amigable donde el usuario puede ingresar una secuencia de referencias de páginas y el tamaño de los marcos. La interfaz incluye:

Entrada de datos: Campos para ingresar la secuencia de referencias de páginas y el tamaño de los marcos.

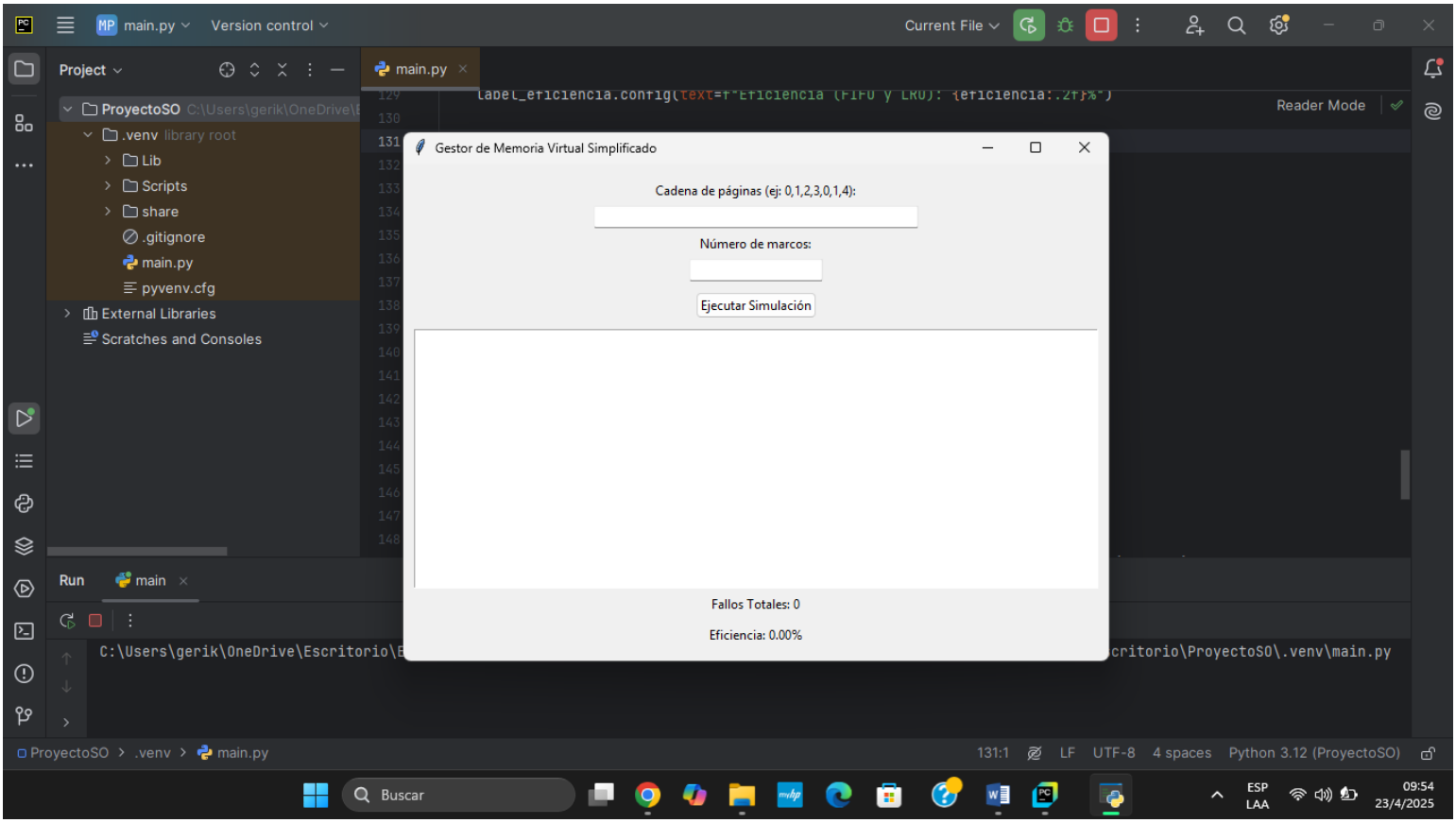
Botón de ejecución: Inicia la simulación de los algoritmos.

Área de resultados: Muestra los resultados de la simulación en formato tabular. Aquí en el área e resultados decidi que aparecieran los tres algoritmos ya ejecutados, esto para poder ver las ventajas y desventajas que tiene cada uno de los algoritmos, así como para poder ver los fallos de página de cada uno y así poder compararlos.

Mostrare una imagen del código de la interfaz y de la interfaz en si para que se pueda

```
132  # Ventana principal de la aplicación
133  root = tk.Tk()
134  root.title("Gestor de Memoria Virtual Simplificado")
135
136  frame = ttk.Frame(root, padding="10")
137  frame.pack(fill="both", expand=True)
138
139  # Etiquetas y campos de entrada
140  ttk.Label(frame, text="Cadena de páginas (ej: 0,1,2,3,0,1,4):").pack(pady=5)
141  entrada_paginas = ttk.Entry(frame, width=50)
142  entrada_paginas.pack()
143
144  ttk.Label(frame, text="Número de marcos:").pack(pady=5)
145  entry_marcos = ttk.Entry(frame)
146  entry_marcos.pack()
147
148  # Botón para ejecutar la simulación
149  ttk.Button(frame, text="Ejecutar Simulación", command=ejecutar_algoritmos).pack(pady=10)
150
151  # Cuadro de texto para resultados
152  text_resultado = tk.Text(frame, wrap="word", height=15, width=80)
153  text_resultado.pack(fill="both", expand=True)
154  text_resultado.config(state=tk.DISABLED, font=("Courier", 10))
155
156  # Etiquetas para mostrar las estadísticas
```

Interfaz del programa



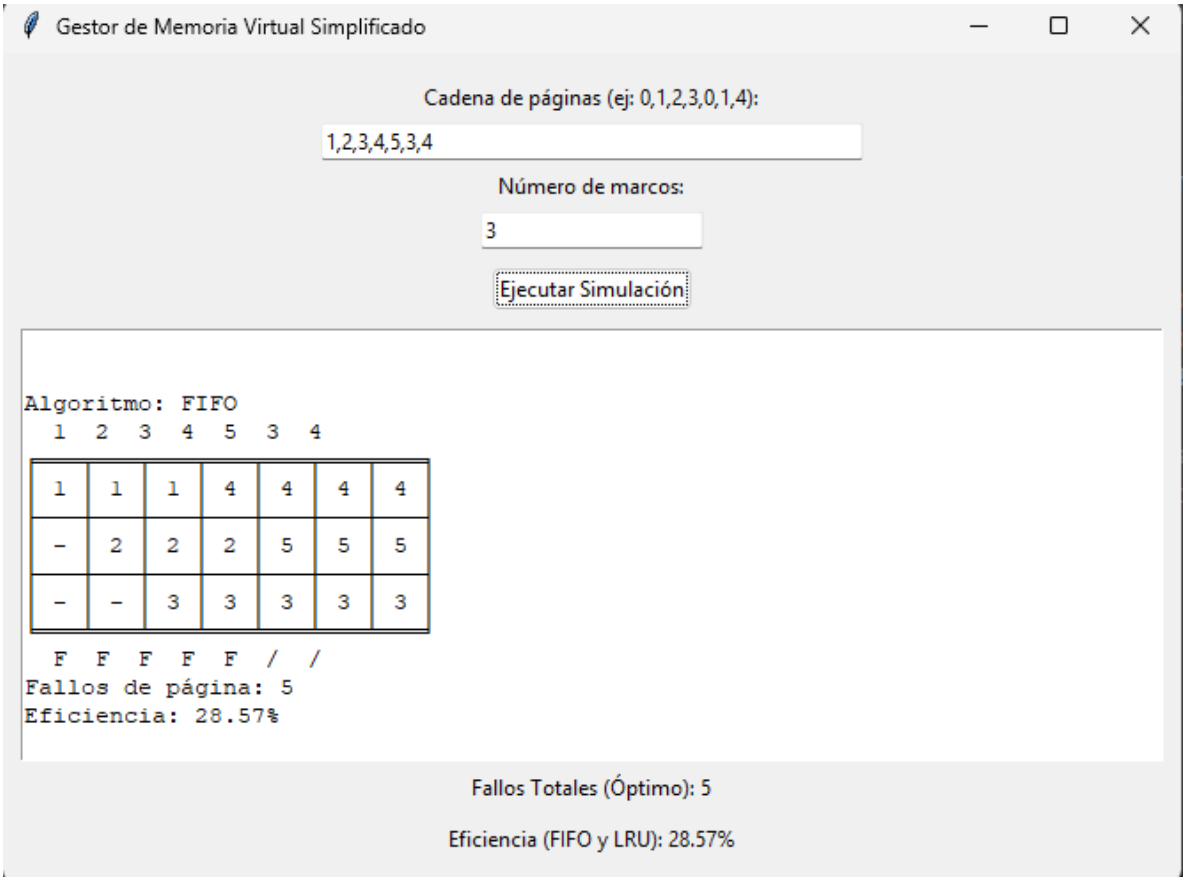
Resultados de la Simulación

Los resultados se presentan en tablas que muestran el estado de los marcos de página en cada paso de la simulación. Además, se indica la cantidad de fallos de página para cada algoritmo.

A continuación, mostrare el resultado de la simulación de cada algoritmo, con su respectiva explicación.

Algoritmo FIFO

Con una cadena de entrada de (1,2,3,4,5,3,4) y un tamaño de marcos de 3.



Referencia a la página 1: Estado inicial: No hay páginas en memoria. Acción: La página 1 se carga en el primer marco. Fallo de página: Sí. Memoria: [1, -, -].

Referencia a la página 2: Acción: La página 2 se carga en el segundo marco. Fallo de página: Sí. Memoria: [1, 2, -].

Referencia a la página 3: Acción: La página 3 se carga en el tercer marco. Fallo de página: Sí. Memoria: [1, 2, 3].

Referencia a la página 4: Acción: No hay marcos libres. Se reemplaza la página más antigua (página 1). Fallo de página: Sí. Memoria: [4, 2, 3]

Referencia a la página 5: Acción: Se reemplaza la siguiente página más antigua (página 2). Fallo de página: Sí. Memoria: [4, 5, 3].

Referencia a la página 3: Acción: La página 3 ya está en memoria. Fallo de página: No. Memoria: [4, 5, 3].

Referencia a la página 4: Acción: La página 4 ya está en memoria. Fallo de página: No. Memoria: [4, 5, 3].

Al final se contabilizan 5 fallos de página, con una eficiencia del 28.57% lo que nos indica que solo 28,57% de las veces evitamos 1 fallo de página.

Algoritmo LRU

Con una cadena de entrada de (1,2,3,4,5,3,4) y un tamaño de marcos de 3.

Referencia a la página 1: Estado inicial: No hay páginas en memoria. Acción: La página 1 se carga en el primer marco. Fallo de página: Sí. Memoria: [1, -, -].

Referencia a la página 2: Acción: La página 2 se carga en el segundo marco. Fallo de página: Sí. Memoria: [1, 2, -].

Referencia a la página 3: Acción: La página 3 se carga en el tercer marco. Fallo de página: Sí. Memoria: [1, 2, 3].

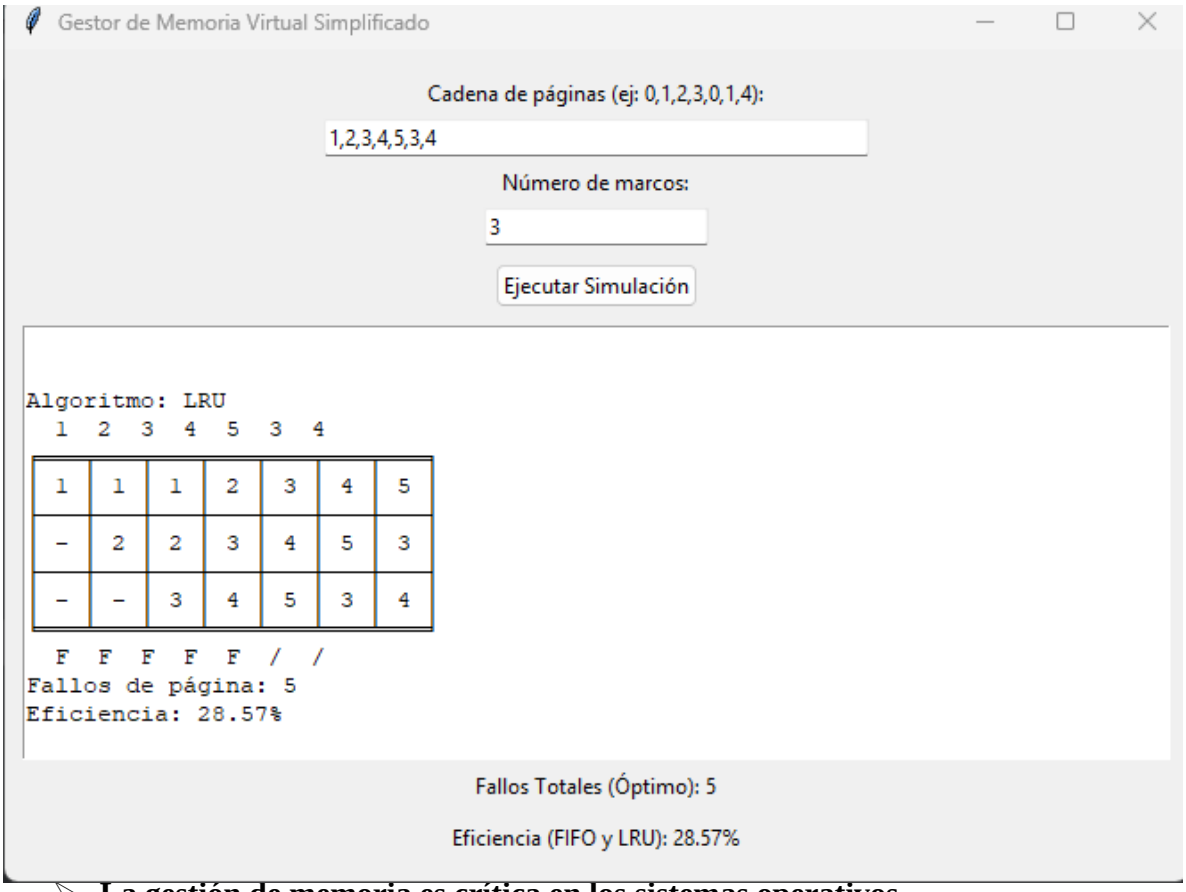
Referencia a la página 4: Acción: No hay marcos libres. Se reemplaza la página menos recientemente utilizada (página 1). Fallo de página: Sí. Memoria: [4, 2, 3].

Referencia a la página 5: Acción: Se reemplaza la siguiente página menos recientemente utilizada (página 2). Fallo de página: Sí. Memoria: [4, 5, 3].

Referencia a la página 3: Acción: La página 3 ya está en memoria. Fallo de página: No. Memoria: [4, 5, 3].

Referencia a la página 4: Acción: La página 4 ya está en memoria. Fallo de página: No. Memoria: [4, 5, 3].

Al final se contabilizan 5 fallos de páginas, con una eficiencia del 28.57% lo que nos indica que solo 28,57% de las veces evitamos 1 fallo de página.



- **La gestión de memoria es crítica en los sistemas operativos.**

  Este proyecto demuestra cómo la división de la memoria en marcos y de los procesos en páginas permite simular el comportamiento real de un sistema con memoria virtual, donde no siempre se puede tener todo cargado en RAM.

- **FIFO es simple, pero no siempre eficiente.**

  El algoritmo FIFO (First In, First Out) reemplaza la página más antigua, sin

importar si todavía se necesita. Esto puede provocar más fallos de página en comparación con otros métodos más inteligentes.

- **LRU mejora el uso de memoria.**

  LRU (Least Recently Used) intenta mantener en memoria las páginas que se han usado recientemente, asumiendo que probablemente se usarán de nuevo pronto. Esto generalmente reduce la cantidad de fallos de página, aunque requiere más seguimiento del uso de cada página.

- **El algoritmo óptimo sirve como punto de comparación.**

  Aunque no es posible implementarlo en la vida real porque necesita conocer el futuro de las referencias de páginas, el algoritmo óptimo permite establecer una base teórica para evaluar la eficiencia de otros algoritmos.

Lo que intente dar a trasmitir con este proyecto es demostrar cómo diferentes algoritmos de sustitución de páginas pueden ser implementados y comparados, proporcionando una visión clara de su funcionamiento y eficiencia relativa.