

UNIVERSIDAD AUTONOMA DE TAMAULIPAS

FACULTAD DE INGENIERIA TAMPICO

SISTEMAS OPERATIVOS

11:00 am – 12:00 pm

INTEGRANTES DEL EQUIPO:

- Arguelles Obregón René
- Rocha Coronado Carlos Julián
- Rodríguez Hernández Juan Julián
- Terán Ramírez Leonardo Alonso

SEMESTRE: 6° GRUPO: J

CARRERA: ING. EN SISTEMAS COMPUTACIONALES

PROFESOR: MUÑOZ QUINTERO DANTE ADOLFO

CICLO ESCOLAR: OTOÑO 2025-3

Producto Integrador

Manual de Usuario del Programa

Índice

Índice.....	2
Especificación del Proyecto.....	3
Requisitos de Instalación y Ejecución.....	4
Configuración del Sistema.....	6
Guía de Operación.....	7
Interpretación de la Interfaz Visual.....	9
Código Fuente del Simulador.....	10
Capturas de Pantalla del Simulador.....	43

Especificación del Proyecto

El objetivo principal de este software es la simulación académica de un gestor de memoria en un sistema operativo multiprogramado. El sistema emula el comportamiento de la unidad de gestión de memoria (MMU) utilizando la técnica de paginación, permitiendo visualizar cómo se traducen las direcciones lógicas de los procesos a direcciones físicas en la RAM.

Además, el simulador integra un mecanismo de Memoria Virtual (Swapping), el cual se activa cuando la memoria física se agota, permitiendo evaluar el rendimiento de distintos algoritmos de reemplazo de páginas (FIFO, LRU, Clock). Esto permite al usuario comprender visualmente los conceptos de:

- Asignación dinámica de recursos.
- Ciclo de vida de los procesos (Nuevo, Listo, Ejecutando, Bloqueado, Terminado).
- Gestión de fallos de página (Page Faults) y el intercambio de marcos entre RAM y Disco (Swap).

Requisitos de Instalación y Ejecución

Requisitos Previos del Sistema

Dado que el simulador ha sido desarrollado en lenguaje Python, es necesario contar con un entorno compatible. El software no requiere librerías externas pesadas, ya que utiliza la biblioteca estándar de Python.

- **Intérprete:** Python 3.6 o superior.
- **Sistema Operativo:** Multiplataforma (Windows, Linux, macOS).
- **Librerías Estándar:** El script hace uso de configparser, math, os, random, time, y collections. Estas vienen instaladas por defecto con Python.

Archivos Necesarios

Para la correcta ejecución, asegúrese de tener en el mismo directorio:

- **simulador_memoria.py:** El script principal que contiene toda la lógica del Kernel simulado y la interfaz CLI.
- **config.ini:** Archivo de configuración que define los parámetros del hardware simulado. **Nota:** Si este archivo no existe, el programa lo generará automáticamente con valores por defecto en la primera ejecución.

Instrucciones de Ejecución

El programa se ejecuta a través de la interfaz de línea de comandos (Terminal o CMD). Siga estos pasos:

1. Abra su terminal o símbolo del sistema.
2. Navegue hasta la carpeta donde descargó el archivo .py.
3. Ejecute el siguiente comando:

python simulador_memoria.py

(Dependiendo de su instalación, el comando podría ser python3 simulador_memoria.py)

Configuración del Sistema

El comportamiento del hardware simulado se controla mediante el archivo config.ini. A continuación se detallan los parámetros permitidos bajo la sección [SYSTEM]:

- **ram_kb:** Define el tamaño total de la Memoria RAM física en Kilobytes. Por ejemplo: 2048 (2 MB).
- **swap_kb:** Define el tamaño total del Área de Intercambio (espacio en disco) en Kilobytes. Por ejemplo: 4096 (4 MB).
- **page_kb:** Tamaño de cada página (espacio lógico) y marco (espacio físico) en Kilobytes. Este valor determina cuántos marcos totales estarán disponibles dividiendo ram_kb / page_kb. Por ejemplo: 256.
- **replacement:** Define el algoritmo de reemplazo de páginas a utilizar cuando la RAM está llena. Los valores aceptados son:
 - **FIFO:** (First-In, First-Out) Expulsa la página que llegó primero.
 - **LRU:** (Least Recently Used) Expulsa la página menos usada recientemente.
 - **CLOCK:** Algoritmo de reloj (Segunda oportunidad).
- **tlb_enabled:** (True/False) Activa o desactiva la simulación del Translation Lookaside Buffer (memoria caché de paginación).
- **arrival_prob:** Valor decimal (0.0 a 1.0) que representa la probabilidad de que llegue un proceso nuevo aleatorio en cada ciclo de reloj.

Guía de Operación

El simulador opera mediante un menú interactivo numérico. A continuación se describe la funcionalidad técnica de cada opción:

1. **Crear Proceso (Manual):** Permite al usuario injectar un proceso específico al sistema. Se solicitarán:
 - **Nombre:** Identificador legible.
 - **Memoria (KB):** Tamaño total del proceso. El sistema calculará automáticamente cuántas páginas necesita (Redondeo hacia arriba: techos(memoria / tamaño_pagina)).
 - **Instrucciones:** Cantidad de ciclos de CPU que el proceso necesita para terminar.
2. **Avanzar Ciclo de Reloj (Tick del CPU):** El simulador funciona por eventos discretos. Esta opción avanza el reloj del sistema (Ticks += 1). En cada tick ocurren tres cosas:
 - El CPU ejecuta una instrucción del proceso actual.
 - Se verifica la probabilidad de llegada de un nuevo proceso aleatorio.
 - Se actualizan los tiempos de referencia para los algoritmos LRU.
3. **Suspender Proceso:** Mueve un proceso del estado LISTO o EJECUTANDO a la cola de BLOQUEADO (ESPERANDO). Útil para simular espera de E/S. Se requiere ingresar el PID.
4. **Reanudar Proceso:** Mueve un proceso de la cola de BLOQUEADO de regreso a la cola de LISTOS, permitiendo que el planificador lo asigne al CPU nuevamente.
5. **Forzar Terminación:** Elimina un proceso del sistema inmediatamente, liberando todos sus marcos de memoria RAM y slots de Swap asociados.
6. **Demo (Generación Aleatoria):** Fuerza la creación inmediata de un proceso con parámetros aleatorios (tamaño y duración) para pruebas de estrés rápidas.

7. **Ver Historial:** Despliega el log de eventos del sistema, útil para auditar cuándo ocurrieron los Fallos de Página (Page Faults), operaciones de Swap-In/Swap-Out y terminación de procesos.
8. **Mostrar Métricas Detalladas:** Muestra estadísticas acumuladas de rendimiento: Total de accesos a memoria, cantidad total de fallos, conteo de operaciones de intercambio y porcentaje de utilización de la RAM.

Interpretación de la Interfaz Visual

El simulador ofrece una visualización en tiempo real del mapa de memoria utilizando nomenclatura basada en texto.

Lectura de Mapas de Memoria (RAM): La sección "MAPA RAM" muestra una cuadrícula donde cada celda representa un marco físico.

- **[M{i}]: Libre:** Indica que el Marco número i está vacío y disponible.
- **[M{i}]: PID{x}, Pag{y}, Ia:{t}]:** Indica que el Marco i está ocupado por:
 - **PID:** El ID del proceso dueño.
 - **Pag:** El número de página lógica del proceso almacenada ahí.
 - **Ia:** (Last Access) El momento del último acceso (dato usado para el algoritmo LRU).

Cómo identificar un Swap: Existen dos formas visuales de identificar que ha ocurrido un intercambio (Swapping):

1. En el MAPA SWAP: Verifique la sección inferior de la pantalla. Si aparece una celda como **[S{i}]: PID{x}...]**, significa que esa página fue expulsada de la RAM y escrita en el disco (**slot S{i}**).
2. En la Tabla de Páginas: Observe la lista "TABLAS DE PAGINAS". Si una entrada dice **idx->S{i}(Swap)**, indica que la página lógica **idx** no está presente en memoria física, sino en el área de intercambio. Intentar acceder a ella provocará un "Fallo de Página".

Código Fuente del Simulador

A continuación, se presenta el código fuente completo del simulador desarrollado en Python (**simulador_memoria.py**). Este script integra todas las clases descritas en el manual técnico (PCB, SimuladorOS, TLB) y contiene la lógica principal de paginación y algoritmos de reemplazo.

```
#!/usr/bin/env python3

# simulador_memoria.py
```

```
"""

Simulador funcional de Gestor de Memoria RAM y Swap con paginación.
```

Cumple requisitos:

- Lectura de config.ini: memoria RAM, swap, tamaño de página, algoritmo reemplazo, prob. llegada dinámica, TLB opcional.
- Gestión dinámica de procesos (creación manual o llegada aleatoria).
- Cálculo de páginas por proceso, tablas de páginas, mapa de marcos RAM + Swap.
- Swapping con algoritmos: FIFO, LRU, CLOCK.
- Visualización CLI del estado y métricas (fallos de página, utilización).
- Registros de operaciones de swapping y fallos de página.

```
"""

import configparser

import math

import os

import random

import time
```

```
from collections import deque, OrderedDict

# -----
# Configuración por defecto (se crea config.ini si no existe)
# -----


DEFAULT_CONFIG = {

    "SYSTEM": {

        "ram_kb": "2048",      # Tamaño RAM en KB

        "swap_kb": "4096",     # Tamaño Swap en KB

        "page_kb": "256",      # Tamaño de página/marco en KB

        "replacement": "FIFO", # FIFO | LRU | CLOCK

        "tlb_enabled": "False", # True|False (opcional)

        "tlb_size": "4",       # entradas TLB si está activa

        "arrival_prob": "0.25", # probabilidad por tick de llegada aleatoria de
                               # proceso

        "max_random_proc_mem_kb": "1024", # max memoria a solicitar proceso
                                         # aleatorio

        "max_random_instr": "30"   # max instrucciones (ciclos) para proceso
                                 # aleatorio

    }

}

# -----
# UTIL: crear o leer config
# -----
```

```

CONFIG_FILE = "config.ini"

def ensure_config():

    if not os.path.exists(CONFIG_FILE):

        cfg = configparser.ConfigParser()

        cfg["SYSTEM"] = DEFAULT_CONFIG["SYSTEM"]

        with open(CONFIG_FILE, "w") as f:

            cfg.write(f)

        print(f"Se generó archivo de configuración por defecto: {CONFIG_FILE}")

    config = configparser.ConfigParser()

    config.read(CONFIG_FILE)

    # validar y devolver valores

    s = config["SYSTEM"]

    return {

        "ram_kb": int(s.get("ram_kb", DEFAULT_CONFIG["SYSTEM"]["ram_kb"])),


        "swap_kb": int(s.get("swap_kb",
        DEFAULT_CONFIG["SYSTEM"]["swap_kb"])),


        "page_kb": int(s.get("page_kb",
        DEFAULT_CONFIG["SYSTEM"]["page_kb"])),


        "replacement": s.get("replacement",
        DEFAULT_CONFIG["SYSTEM"]["replacement"]).upper(),


        "tlb_enabled": s.get("tlb_enabled",
        DEFAULT_CONFIG["SYSTEM"]["tlb_enabled"]).lower() in ("1", "true", "yes"),


        "tlb_size": int(s.get("tlb_size", DEFAULT_CONFIG["SYSTEM"]["tlb_size"])),


        "arrival_prob": float(s.get("arrival_prob",
        DEFAULT_CONFIG["SYSTEM"]["arrival_prob"])),

    }

```

```

    "max_random_proc_mem_kb":      int(s.get("max_random_proc_mem_kb",
DEFAULT_CONFIG["SYSTEM"]["max_random_proc_mem_kb"])),


    "max_random_instr":           int(s.get("max_random_instr",
DEFAULT_CONFIG["SYSTEM"]["max_random_instr"]))



}

# -----



# CLASE PCB



# -----



class PCB:



def __init__(self, pid, nombre, tam_kb, prioridad, instrucciones_totales):



    self.pid = pid



    self.nombre = nombre



    self.tamano_kb = tam_kb



    self.prioridad = prioridad



    self.instrucciones_totales = instrucciones_totales



    self.instrucciones_restantes = instrucciones_totales



    self.estado = "NUEVO" # NUEVO, LISTO, EJECUTANDO, ESPERANDO,
TERMINADO, INTERCAMBIADO



# paginación:



    self.paginas = math.ceil(tam_kb / SimuladorOS.page_kb_global) # calculado al agregar al simulador



    # tabla de páginas: cada entrada -> dict {present:bool, frame: int|None,
swap_slot: int|None, referenced:int(for LRU), loaded_time:int}



    self.tabla_paginas = {i: {"present": False, "frame": None, "swap_slot": None,
"last_access": 0} for i in range(self.paginas)}

```

```

self.motivo_terminacion = ""

def __str__(self):

    return f"[PID:{self.pid}] {self.nombre} | Estado:{self.estado}\n"
    "Mem:{self.tamano_kb}KB | Pags:{self.paginas}\n"
    "Rest:{self.instrucciones_restantes}" |



# -----



# TLB simple (opcional)

# -----



class TLB:

    def __init__(self, size):

        self.size = size

        # OrderedDict para LRU por acceso

        self.entries = OrderedDict() # key: (pid,page) -> frame



def lookup(self, pid, page):

    key = (pid, page)

    if key in self.entries:

        # mover al final (reciente)

        frame = self.entries.pop(key)

        self.entries[key] = frame

    return frame

    return None

```

```

def add(self, pid, page, frame):
    key = (pid, page)

    if key in self.entries:
        self.entries.pop(key)

    elif len(self.entries) >= self.size:
        self.entries.popitem(last=False) # evict oldest

    self.entries[key] = frame


def invalidate_pid(self, pid):
    keys = [k for k in self.entries.keys() if k[0] == pid]

    for k in keys:
        self.entries.pop(k)


def __str__(self):
    return str(list(self.entries.items()))


# -----
# SIMULADOR
# -----


class SimuladorOS:

    # variable global para que PCB calcule paginas antes de instancia? la
    # inicializamos al crear simulador.

    page_kb_global = 256

```

```

def __init__(self, ram_kb, swap_kb, page_kb, replacement_algo="FIFO",
tlb_enabled=False, tlb_size=4, arrival_prob=0.25, max_rand_mem=1024,
max_rand_instr=30):

    # parámetros del sistema

    SimuladorOS.page_kb_global = page_kb

    self.ram_kb = ram_kb

    self.swap_kb = swap_kb

    self.page_kb = page_kb

    self.replacement_algo = replacement_algo.upper()

    self.tlb_enabled = tlb_enabled

    self.arrival_prob = arrival_prob

    self.max_rand_mem = max_rand_mem

    self.max_rand_instr = max_rand_instr


    # marcos

    self.num_marcos_ram = ram_kb // page_kb

    self.num_marcos_swap = swap_kb // page_kb


    # estructuras físicas

    # RAM: lista de marcos; cada marco: None o dict {pid, page, loaded_time,
last_access}

    self.ram = [None for _ in range(self.num_marcos_ram)]

    self.free_frames = deque(range(self.num_marcos_ram)) # frames libres

    # Swap: lista de slots; each slot: None or dict {pid,page,stored_time}

    self.swap = [None for _ in range(self.num_marcos_swap)]

```

```
self.free_swap_slots = deque(range(self.num_marcos_swap))

# replacement metadata

self fifo queue = deque() # frames en orden de carga (para FIFO)

# for LRU we use last_access stored in frame dict; for CLOCK implement
circular pointer

self.clock_pointer = 0

# procesos y colas

self.cola_listos = deque()

self.cola_bloqueados = deque()

self.procesos_terminados = []

self.proceso_actual = None

self.contador_pid = 1

# TLB

self.tlb = TLB(tlb_size) if tlb_enabled else None

# métricas

self.total_accesos = 0

self.total_fallos = 0

self.swap_ins = 0

self.swap_outs = 0

self.ticks = 0
```

```

self.log_events = [] # lista de strings con eventos (swaps, fallos, etc.)

# reloj lógico para LRU timestamp
self.access_clock = 0

# -----
# UTIL y LOG
# -----


def log(self, s):
    t = f"[t={self.ticks}] {s}"
    print(t)
    self.log_events.append(t)

# -----


def map_utilization(self):
    used = sum(1 for f in self.ram if f is not None)

    return used, self.num_marcos_ram, used / self.num_marcos_ram if self.num_marcos_ram > 0 else 0.0

# -----


# CREAR PROCESO (manual o aleatorio)
# -----


def crear_proceso(self, nombre, tam_kb, prioridad, instr):
    pid = self.contador_pid
    # instanciar PCB con paginas calculadas

```

```

pcb = PCB(pid, nombre, tam_kb, prioridad, instr)

self.contador_pid += 1

# marcar y colocar en lista de listos; la asignación de páginas se hace aquí
(cargar en RAM o Swap)

pcb.estado = "LISTO"

self.cola_listos.append(pcb)

self.log(f" ✅ Proceso creado: {pcb}")

# intentar asignar sus páginas (carga inicial parcial: intentamos traer todas
las páginas si hay marcos, si no, usar swapping)

self.allocate_pages_for_process(pcb)

return pcb


def crear_proceso_aleatorio(self):

    # generar nombre y parámetros

    tam = random.randint(1, min(max(1, self.max_rand_mem), self.ram_kb +
self.swap_kb)))

    instr = random.randint(1, max(1, self.max_rand_instr))

    nombre = f"P{self.contador_pid}_rand"

    prio = random.randint(1, 10)

    return self.crear_proceso(nombre, tam, prio, instr)


# -----
# ASIGNACIÓN Y PAGINACIÓN
# -----


def allocate_pages_for_process(self, pcb):

```

```
# por cada página lógica intentaremos asignar un marco en RAM, si no hay  
marco libre -> victimizar y swap out
```

```
for pagina in range(pcb.paginas):  
  
    if self.free_frames:  
  
        frame = self.free_frames.popleft()  
  
        self.place_page_in_frame(pcb, pagina, frame)  
  
    else:  
  
        # no hay marcos libres -> realizar swapping para obtener un marco  
  
        victim_frame = self.select_victim_frame()  
  
        if victim_frame is None:  
  
            # Swap lleno -> forzar almacenamiento directo en swap si hay slots  
  
            if self.free_swap_slots:  
  
                swap_slot = self.free_swap_slots.popleft()  
  
                # asignamos la página directamente a swap (no ocupa RAM)  
  
                pcb.tabla_paginas[pagina]["present"] = False  
  
                pcb.tabla_paginas[pagina]["frame"] = None  
  
                pcb.tabla_paginas[pagina]["swap_slot"] = swap_slot  
  
                self.swap[swap_slot] = {"pid": pcb.pid, "page": pagina,  
"stored_time": self.ticks}  
  
                self.log(f"🕒 Página {pagina} de {pcb.nombre} (PID {pcb.pid})  
colocada directamente en Swap, slot {swap_slot} (RAM llena y swap con  
espacio).")  
  
                self.swap_outs += 1  
  
            else:  
  
                # Swap y RAM llenos -> no hay dónde colocar. Dejamos la página  
                marcada sin ubicación (se consideraría error o espera)
```

```
        self.log(f"❌ Espacio insuficiente: sin marcos y swap lleno. Página {pagina} de {pcb.nombre} no asignada.")
```

```
else:
```

```
    # expulsar la página víctima hacia swap (o liberar marco si el contenido ya estaba en swap? en nuestro modelo movemos)
```

```
    self.swap_out_frame(victim_frame)
```

```
    # ahora usar ese marco liberado
```

```
    self.place_page_in_frame(pcb, pagina, victim_frame)
```

```
def place_page_in_frame(self, pcb, pagina, frame_index):
```

```
    # colocar la página en RAM marco frame_index
```

```
    self.ram[frame_index] = {"pid": pcb.pid, "page": pagina, "loaded_time": self.ticks, "last_access": self.ticks, "referenced": True}
```

```
    pcb.tabla_paginas[pagina]["present"] = True
```

```
    pcb.tabla_paginas[pagina]["frame"] = frame_index
```

```
    pcb.tabla_paginas[pagina]["swap_slot"] = None
```

```
    pcb.tabla_paginas[pagina]["last_access"] = self.ticks
```

```
    # actualizar estructuras de reemplazo
```

```
    if self.replacement_algo == "FIFO":
```

```
        self fifo_queue.append(frame_index)
```

```
        # NOTE: for LRU we will update last_access on accesses; for CLOCK we utilize 'referenced' bit stored in frame
```

```
        self.log(f"🕒 Cargada Página {pagina} de Proceso {pcb.nombre} (PID {pcb.pid}) en Marco RAM {frame_index}.")
```

```
def swap_out_frame(self, frame_index):
```

```

# mover contenido del marco a swap (o a un slot libre)

content = self.ram[frame_index]

if content is None:

    return

pid = content["pid"]

page = content["page"]

# buscar slot de swap libre

if not self.free_swap_slots:

    # si no hay swap libre, debemos evictar alguna entrada de swap o fallar
    (aquí elegimos forzar reemplazo FIFO en swap: sobrescribir el primero)

    # pero para simplicidad: hacemos búsqueda lineal y si no hay slot libre
    devolvemos None

    self.log("⚠ Swap lleno: no se pudo mover página (modelo simple).")

    return None

slot = self.free_swap_slots.popleft()

self.swap[slot] = {"pid": pid, "page": page, "stored_time": self.ticks}

# actualizar tabla de páginas del proceso víctima

victim_pcb = self.find_pcb_by_pid(pid)

if victim_pcb:

    victim_pcb.tabla_paginas[page]["present"] = False

    victim_pcb.tabla_paginas[page]["frame"] = None

    victim_pcb.tabla_paginas[page]["swap_slot"] = slot

    victim_pcb.tabla_paginas[page]["last_access"] = self.ticks

# limpiar marco RAM

```

```

    self.ram[frame_index] = None

    # actualizar estructuras de replacement: quitar frame de FIFO si está ahí

    try:

        self fifo _queue.remove(frame_index)

    except ValueError:

        pass

    # añadir marco liberado a free_frames (quedará disponible)

    self.free_frames.append(frame_index)

    self.swap_outs += 1

    self.log(f"🕒 Swapping: Página {page} de Proceso PID {pid} movida a Swap
slot {slot} desde marco {frame_index}.")
```

return slot

```

def select_victim_frame(self):

    # elegir frame víctima según algoritmo

    if self.replacement_algo == "FIFO":

        # FIFO: el primer frame en cola FIFO

        if self fifo _queue:

            return self fifo _queue.popleft()

        else:

            return None

    elif self.replacement_algo == "LRU":

        # elegir frame con menor last_access

        oldest = None
```

```

oldest_time = None

for i, f in enumerate(self.ram):

    if f is None:

        continue

    la = f.get("last_access", 0)

    if oldest is None or la < oldest_time:

        oldest = i

        oldest_time = la

return oldest

elif self.replacement_algo == "CLOCK":

    # algoritmo reloj simple: iterar buscando referenced == False, si
    referenced True -> clear and advance. Usamos pointer circular.

    n = self.num_marcos_ram

    scans = 0

    while scans < n:

        fr = self.clock_pointer % n

        f = self.ram[fr]

        if f is None:

            # marco libre: utilizarlo (no victim)

            self.clock_pointer = (self.clock_pointer + 1) % n

            return fr

        if not f.get("referenced", False):

            victim = fr

            self.clock_pointer = (fr + 1) % n

```

```

        return victim

    else:

        # quitar bit referenced y continuar

        f["referenced"] = False

        self.clock_pointer = (self.clock_pointer + 1) % n

        scans += 1

        # si no encontramos, forzamos victim en pointer

        victim = self.clock_pointer % n

        self.clock_pointer = (victim + 1) % n

        return victim

else:

    # default FIFO

    if self fifo_queue:

        return self fifo_queue.popleft()

    return None

# -----
# ACCESO A PÁGINAS (lectura/ejecución) - simula referencia a una dirección
# -----


def access_page(self, pcb, page_index):

    """



Simula acceso a una página: si está en RAM -> hit; si no -> page fault ->
traer desde swap o asignar marco

"""

```

```

    self.total_accesos += 1

    self.access_clock += 1

    self.ticks += 1 # cada acceso avanza el tiempo

    # TLB lookup

    if self.tlb:

        frame = self.tlb.lookup(pcb.pid, page_index)

        if frame is not None:

            # hit TLB

            # actualizar metadata LRU

            if self.ram[frame]:

                self.ram[frame]["last_access"] = self.access_clock

                self.ram[frame]["referenced"] = True

                pcb.tabla_paginas[page_index]["last_access"] = self.access_clock

            return True # hit

    # page table

    entry = pcb.tabla_paginas.get(page_index)

    if entry is None:

        self.log(f"❌ Acceso inválido: página {page_index} fuera del rango para  
PID {pcb.pid}.")"

        return False

    if entry["present"]:

        # hit

        frame = entry["frame"]

        # actualizar last_access (LRU) y referenced (Clock)

```

```

if frame is not None and self.ram[frame] is not None:

    self.ram[frame]["last_access"] = self.access_clock

    self.ram[frame]["referenced"] = True

    entry["last_access"] = self.access_clock

    # actualizar TLB

    if self.tlb:

        self.tlb.add(pcb.pid, page_index, frame)

    return True

else:

    # page fault

    self.total_fallos += 1

    self.log(f"⚠ Fallo de página: Proceso {pcb.nombre} PID {pcb.pid} -> Página {page_index} no en RAM.")

    # si está en swap -> traer desde swap

    if entry["swap_slot"] is not None:

        slot = entry["swap_slot"]

        # si hay frame libre usarlo

        if self.free_frames:

            frame = self.free_frames.popleft()

        else:

            frame = self.select_victim_frame()

        if frame is None:

            self.log("❌ No se pudo obtener marco para traer de swap (swap lleno y RAM sin victim).")

```

```

        return False

    # swap out victim

    self.swap_out_frame(frame)

    # after swap_out_frame, frame moved to free_frames; remove one

    if self.free_frames:

        frame = self.free_frames.popleft()

    # mover de swap a frame

        self.ram[frame] = {"pid": pcb.pid, "page": page_index, "loaded_time": self.ticks, "last_access": self.access_clock, "referenced": True}

        pcb.tabla_paginas[page_index]["present"] = True

        pcb.tabla_paginas[page_index]["frame"] = frame

        pcb.tabla_paginas[page_index]["swap_slot"] = None

        pcb.tabla_paginas[page_index]["last_access"] = self.access_clock

    # limpiar slot swap

        self.swap[slot] = None

        self.free_swap_slots.append(slot)

        self.swap_ins += 1

        self.log(f"Swap-In: Página {page_index} de PID {pcb.pid} traída desde Swap slot {slot} al marco {frame}.")"

    # replacement meta update

    if self.replacement_algo == "FIFO":

        self fifo_queue.append(frame)

    # actualizar TLB

    if self.tlb:

```

```

        self.tlb.add(pcb.pid, page_index, frame)

    return True

else:

    # página nunca cargada antes (nueva) -> asignar marco si hay; si no ->
    victimizar y colocar

    if self.free_frames:

        frame = self.free_frames.popleft()

        self.place_page_in_frame(pcb, page_index, frame)

        return True

    else:

        victim_frame = self.select_victim_frame()

        if victim_frame is None:

            self.log("X No se pudo obtener marco para cargar nueva
página.")

            return False

        self.swap_out_frame(victim_frame)

        self.place_page_in_frame(pcb, page_index, victim_frame)

        return True

# -----
# BÚSQUEDAS / UTILIDADES
# -----

def find_pcb_by_pid(self, pid):

    # buscar en colas y en proceso_actual y en terminados

```

```
if self.proceso_actual and self.proceso_actual.pid == pid:  
  
    return self.proceso_actual  
  
for p in self.cola_listos:  
  
    if p.pid == pid:  
  
        return p  
  
for p in self.cola_bloqueados:  
  
    if p.pid == pid:  
  
        return p  
  
for p in self.procesos_terminados:  
  
    if p.pid == pid:  
  
        return p  
  
return None
```

```
# -----
```

```
# PLANIFICACIÓN (simple: FCFS o RR)
```

```
# -----
```

```
def planificar(self):
```

```
if not self.proceso_actual and self.cola_listos:
```

```
    self.proceso_actual = self.cola_listos.popleft()
```

```
    self.proceso_actual.estado = "EJECUTANDO"
```

```
        self.log(f"⚡ Despachando a CPU: {self.proceso_actual.nombre} (PID {self.proceso_actual.pid})")
```

```
def ejecutar_ciclo(self):
```

```

self.ticks += 1

self.access_clock += 1

# generación dinámica de llegada

if random.random() < self.arrival_prob:

    self.crear_proceso_aleatorio()

# si no hay proceso en CPU planificar

if not self.proceso_actual:

    self.planificar()

if not self.proceso_actual:

    # CPU ocioso

    return

# simular ejecución: el proceso accede a una página aleatoria de su espacio

p = self.proceso_actual

# elegir página que exista

if p.paginas == 0:

    # proceso sin memoria? finalizamos

    self.terminar_proceso(p, motivo="Sin páginas")

    self.proceso_actual = None

    return

page_to_access = random.randint(0, p.paginas - 1)

```

```

hit = self.access_page(p, page_to_access)

# consumir una instrucción

p.instrucciones_restantes -= 1

# si terminó

if p.instrucciones_restantes <= 0:

    self.terminar_proceso(p, motivo="Finalización Normal")

    self.proceso_actual = None

return

# En modelos RR podríamos rotar, pero dejamos FCFS por simplicidad del
requerimiento

# actualizar referenced bits si es CLOCK ya hecho en access_page

# -----
# TERMINAR / FORZAR / SUSPENDER / REANUDAR
# -----


def terminar_proceso(self, proceso, motivo="Normal"):

    # liberar marcos y slots de swap asociados al proceso

    # liberar marcos RAM

    for page_idx, entry in proceso.tabla_paginas.items():

        if entry["present"] and entry["frame"] is not None:

            fr = entry["frame"]

            # limpiar marco

```

```

if 0 <= fr < self.num_marcos_ram and self.ram[fr] is not None:

    self.ram[fr] = None

    self.free_frames.append(fr)

try:

    self fifo_queue.remove(fr)

except ValueError:

    pass

if entry["swap_slot"] is not None:

    slot = entry["swap_slot"]

    if 0 <= slot < self.num_marcos_swap and self.swap[slot] is not None:

        self.swap[slot] = None

        self.free_swap_slots.append(slot)

# invalidar TLB entradas

if self.tlb:

    self.tlb.invalidate_pid(proceso.pid)

proceso.estado = "TERMINADO"

proceso.motivo_terminacion = motivo

self.procesos_terminados.append(proceso)

    self.log(f"🏁 Proceso {proceso.nombre} (PID {proceso.pid}) terminado.
Motivo: {motivo}")

def forzar_terminacion(self, pid_a_matar):

    # buscar y terminar

    for p in list(self.cola_listos):

```

```

if p.pid == pid_a_matar:

    self.cola_listos.remove(p)

    self.terminar_proceso(p, "Forzada por Usuario")

    return

for p in list(self.cola_bloqueados):

    if p.pid == pid_a_matar:

        self.cola_bloqueados.remove(p)

        self.terminar_proceso(p, "Forzada por Usuario")

    return

if self.proceso_actual and self.proceso_actual.pid == pid_a_matar:

    self.terminar_proceso(self.proceso_actual, "Forzada por Usuario")

    self.proceso_actual = None

return

self.log("✖ Proceso no encontrado para terminar.")

```

```

def suspender_proceso(self, pid):

    # si es actual

    if self.proceso_actual and self.proceso_actual.pid == pid:

        self.proceso_actual.estado = "ESPERANDO"

        self.cola_bloqueados.append(self.proceso_actual)

        self.log(f"⏸ Suspendido proceso en ejecución: {self.proceso_actual.nombre}")

        self.proceso_actual = None

    return

```

```
for p in list(self.cola_listos):

    if p.pid == pid:

        self.cola_listos.remove(p)

        p.estado = "ESPERANDO"

        self.cola_bloqueados.append(p)

        self.log(f"🔴 Proceso {p.nombre} movido a ESPERANDO")

    return

self.log("🔴 No se pudo suspender: Proceso no encontrado.")
```

```
def reanudar_proceso(self, pid):

    for p in list(self.cola_bloqueados):

        if p.pid == pid:

            self.cola_bloqueados.remove(p)

            p.estado = "LISTO"

            self.cola_listos.append(p)

            self.log(f"🔵 Reanudado proceso {p.nombre}")

    return

self.log("🔴 No se pudo reanudar: Proceso no encontrado en bloqueados.")
```

```
# -----
```

```
# REPORTES Y VISUALIZACIÓN
```

```
# -----
```

```
def mostrar_estado(self):
```

```

print("\n" + "-"*80)

    print(f"| TICKS: {self.ticks} | Algoritmo Reemplazo: {self.replacement_algo} |
RAM: {self.num_marcos_ram*self.page_kb}KB ({self.num_marcos_ram} marcos)
| SWAP: {self.num_marcos_swap*self.page_kb}KB ({self.num_marcos_swap} marcos) |")

print("-"*80)

    print(f"|" CPU (Ejecutando): [{self.proceso_actual.nombre} if
self.proceso_actual else 'OCIOSO'] ")

    print(f"|" Cola LISTOS: {[f'{p.pid}:{p.nombre}' for p in self.cola_listos]}")

    print(f"|" Cola ESPERANDO: {[f'{p.pid}:{p.nombre}' for p in
self.cola_bloqueados]}")

print("-"*80)

# mapa RAM

print("MAPA RAM (marco_index : contenido )")

for i, fr in enumerate(self.ram):

    if fr is None:

        print(f"[M{i}]: Libre]", end=" ")

    else:

        print(f"[M{i}]: PID{fr['pid']}, Pag{fr['page']}, la:{fr['last_access']}]", end=" ")

    if (i+1) % 6 == 0:

        print()

print("\n" + "-"*80)

# mapa Swap

print("MAPA SWAP (slot_index : contenido )")

for i, s in enumerate(self.swap):

    if s is None:

```

```

print(f"[S{i}: Libre]", end=" ")

else:

    print(f"[S{i}: PID{s['pid']}, Pag{s['page']}, st:{s['stored_time']}]", end=" ")

if (i+1) % 6 == 0:

    print()

print("\n" + "-"*80)

# tablas de páginas por proceso

print("TABLAS DE PAGINAS (por proceso)")

procs = []

if self.proceso_actual: procs.append(self.proceso_actual)

procs += list(self.cola_listos)

procs += list(self.cola_bloqueados)

for p in procs:

    entries = []

    for idx, e in p.tabla_paginas.items():

        if e["present"]:

            entries.append(f"{idx}->M{e['frame']}")

        elif e["swap_slot"] is not None:

            entries.append(f"{idx}->S{s['slot']} if False else e['swap_slot']}(Swap)")

        else:

            entries.append(f"{idx}->(Sin asign)")

    print(f"PID {p.pid} {p.nombre}: " + ", ".join(entries))

print("-"*80)

```

```

# TLB

if self.tlb:

    print("TLB (entries):", self.tlb)

# métricas

used, total, util = self.map_utilization()

print(f"METRICAS: Accesos={self.total_accesos}, Fallos={self.total_fallos},
SwapIns={self.swap_ins}, SwapOuts={self.swap_outs}, Util_RAM={used}/{total}
({util*100:.1f}%)")

if self.total_accesos > 0:

    print(f"Tasa de fallos: {self.total_fallos/self.total_accesos*100:.2f}%")

    print("*"*80)

def ver_historial(self):

    print("\n--- HISTORIAL DE EVENTOS ---")

    for e in self.log_events[-200:]:
        print(e)

    input("Presione Enter para continuar...")

# -----
# INTERFAZ CLI - MAIN
# -----


def main():

    print("--- SIMULADOR DE MEMORIA RAM & SWAP (Paginación) ---")

    # asegurar config

```

```

cfg = ensure_config()

# crear simulador con valores del config

sim = SimuladorOS(
    ram_kb=cfg["ram_kb"],
    swap_kb=cfg["swap_kb"],
    page_kb=cfg["page_kb"],
    replacement_algo=cfg["replacement"],
    tlb_enabled=cfg["tlb_enabled"],
    tlb_size=cfg["tlb_size"],
    arrival_prob=cfg["arrival_prob"],
    max_rand_mem=cfg["max_random_proc_mem_kb"],
    max_rand_instr=cfg["max_random_instr"]
)

# Menú principal

while True:

    sim.mostrar_estado()

    print("1. Crear Proceso (Manual)")

    print("2. Avanzar Ciclo de Reloj (Tick del CPU)")

    print("3. Suspender Proceso (Mover a Espera)")

    print("4. Reanudar Proceso (Mover a Listo)")

    print("5. Forzar Terminación de Proceso")

    print("6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)")

```

```
print("7. Ver Historial de eventos")

print("8. Mostrar métricas detalladas")

print("9. Salir")

op = input("Seleccione una opción: ").strip()

if op == "1":

    try:

        nom = input("Nombre del Proceso: ").strip()

        mem = int(input("Memoria requerida (KB): ").strip())

        prio = int(input("Prioridad (1-10): ").strip())

        instr = int(input("Instrucciones (ciclos CPU): ").strip())

        sim.crear_proceso(nom, mem, prio, instr)

    except Exception as e:

        print("Error: ingrese valores válidos.", e)

elif op == "2":

    sim.ejecutar_ciclo()

    time.sleep(0.15)

elif op == "3":

    try:

        pid = int(input("Ingrese PID a suspender: ").strip())

        sim.suspender_proceso(pid)

    except:

        print("PID inválido.")

elif op == "4":
```

```
try:  
  
    pid = int(input("Ingrese PID a reanudar: ").strip())  
  
    sim.reanudar_proceso(pid)  
  
except:  
  
    print("PID inválido.")  
  
elif op == "5":  
  
    try:  
  
        pid = int(input("Ingrese PID a terminar (forzado): ").strip())  
  
        sim.forzar_terminacion(pid)  
  
    except:  
  
        print("PID inválido.")  
  
elif op == "6":  
  
    sim.crear_proceso_aleatorio()  
  
elif op == "7":  
  
    sim.ver_historial()  
  
elif op == "8":  
  
    print("\n--- METRICAS DETALLADAS ---")  
  
    print(f"Ticks: {sim.ticks}")  
  
    print(f"Accesos: {sim.total_accesos}")  
  
    print(f"Fallos de página: {sim.total_fallos}")  
  
    print(f"Swap in: {sim.swap_ins} | Swap out: {sim.swap_outs}")  
  
    used, total, util = sim.map_utilization()  
  
    print(f"Utilización RAM (marcos usados): {used}/{total} ({util*100:.2f}%)")
```

```
if sim.total_accesos:

    print(f"Tasa de fallos: {sim.total_fallos/sim.total_accesos*100:.2f}%")

    input("Presione Enter para volver...")

elif op == "9":

    print("Cerrando simulador...")

    break

else:

    print("Opción inválida. Intente de nuevo.")

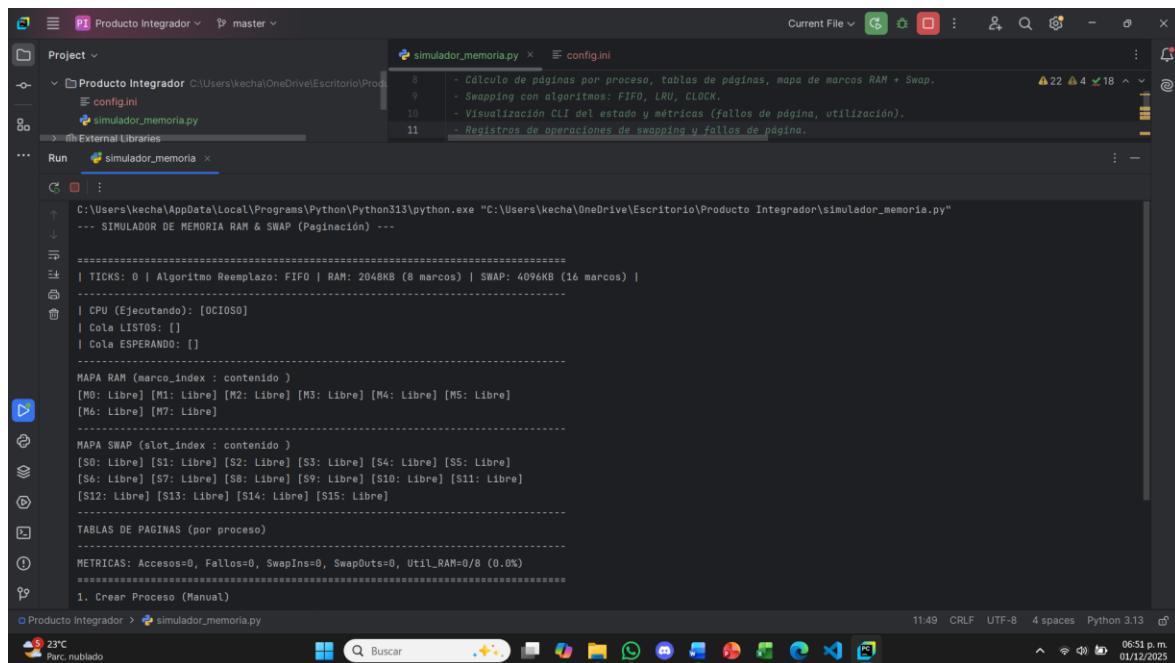
if __name__ == "__main__":

    main()
```

Capturas de Pantalla del Simulador

Inicialización del Sistema

Esta captura documenta el estado inicial del simulador justo después de su ejecución. Se puede validar la correcta lectura del archivo de configuración, mostrando que el sistema ha reservado 2048 KB para la memoria RAM (dividida en 8 marcos) y 4096 KB para el área de intercambio (Swap). Tanto el Mapa de RAM como el de Swap aparecen marcados completamente como [Libre], y las colas de planificación están vacías, lo que confirma que el entorno está limpio y listo para recibir la carga de trabajo sin procesos residuales.



```
Current File: C:\Users\keche\OneDrive\Escritorio\Producto Integrador\simulador_memoria.py
simulador_memoria.py x config.ini
8 - Cálculo de páginas por proceso, tablas de páginas, mapa de marcos RAM + Swap.
9 - Swapping con algoritmos: FIFO, LRU, CLOCK.
10 - Visualización CLI del estado y métricas (fallos de página, utilización).
11 - Registros de operaciones de swapping y fallos de página.

C:\Users\keche\AppData\Local\Programs\Python\Python313\python.exe "C:\Users\keche\OneDrive\Escritorio\Producto Integrador\simulador_memoria.py"
-- SIMULADOR DE MEMORIA RAM & SWAP (Paginación) --

=====
| TICKS: 0 | Algoritmo Reemplazo: FIFO | RAM: 2048KB (8 marcos) | SWAP: 4096KB (16 marcos) |
| CPU (Ejecutando): [0IO50] |
| Cola LISTOS: [] |
| Cola ESPERANDO: [] |

MAPA RAM (marco_index : contenido)
[0: Libre] [1: Libre] [2: Libre] [3: Libre] [4: Libre] [5: Libre]
[6: Libre] [7: Libre]

MAPA SWAP (slot_index : contenido)
[0: Libre] [1: Libre] [2: Libre] [3: Libre] [4: Libre] [5: Libre]
[6: Libre] [7: Libre] [8: Libre] [9: Libre] [10: Libre] [11: Libre]
[12: Libre] [13: Libre] [14: Libre] [15: Libre]

TABLAS DE PAGINAS (por proceso)

METRICAS: Accesos=0, Fallos=0, SwapIns=0, SwapOuts=0, Util_RAM=0/8 (0.0%)
=====

1. Crear Proceso (Manual)
```

Creación Manual de Proceso

En esta imagen se demuestra el uso de la Opción 1 para ingresar un proceso al sistema. El usuario especifica los parámetros operativos: nombre "FIFO", 1024 KB de memoria y prioridad 5. El simulador realiza el cálculo de paginación automáticamente, determinando que se requieren 4 páginas lógicas para satisfacer la demanda de memoria. El log de eventos confirma la transacción exitosa, indicando que las páginas 0, 1, 2 y 3 se han cargado secuencialmente en los marcos 0 a 3 de la RAM.

```

simulador_memoria.py x config.ini
=====
8 - Cálculo de páginas por proceso, tablas de páginas, mapa de marcos RAM + Swap.
9 - Swapping con algoritmos: FIFO, LRU, CLOCK.
10 - Visualización CLI del estado y métricas (fallos de página, utilización).
11 - Registros de operaciones de swapping y fallos de página.

=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos
8. Mostrar métricas detalladas
9. Salir

Seleccione una opción: 1
Nombre del Proceso: FIFO
Memoria requerida (KB): 1024
Prioridad (1-10): 5
Instrucciones (ciclos CPU): 2
[t=0] ✓ Proceso creado: [PID:1] FIFO | Estado:LISTO | Mem:1024KB | Pags:4 | Rest:2
[t=0] ✓ Cargada Página 0 de Proceso FIFO (PID 1) en Marco RAM 0.
[t=0] ✓ Cargada Página 1 de Proceso FIFO (PID 1) en Marco RAM 1.
[t=0] ✓ Cargada Página 2 de Proceso FIFO (PID 1) en Marco RAM 2.
[t=0] ✓ Cargada Página 3 de Proceso FIFO (PID 1) en Marco RAM 3.

=====
| TICKS: 0 | Algoritmo Remplazo: FIFO | RAM: 2048KB (8 marcos) | SWAP: 4096KB (16 marcos) |
=====

11:49 CRLF UTF-8 4 spaces Python 3.13
23°C Parc. nublado Buscar 0652 p.m. 01/12/2025

```

Estado de Memoria y Tablas

Esta captura ofrece una vista detallada de la asignación de recursos resultante de la creación del proceso anterior. En el Mapa RAM, se observa que los primeros cuatro marcos (M0-M3) han cambiado su estado a ocupado por el PID 1, mientras que los marcos M4-M7 permanecen libres. Simultáneamente, la sección de "Tablas de Páginas" valida la traducción de direcciones, mapeando cada página lógica a su marco físico correspondiente. Las métricas inferiores corroboran una utilización del 50% de la memoria principal.

```

simulador_memoria.py x config.ini
=====
11 - Registros de operaciones de swapping y fallos de página.
12 -----
13 import configparser
14 import math

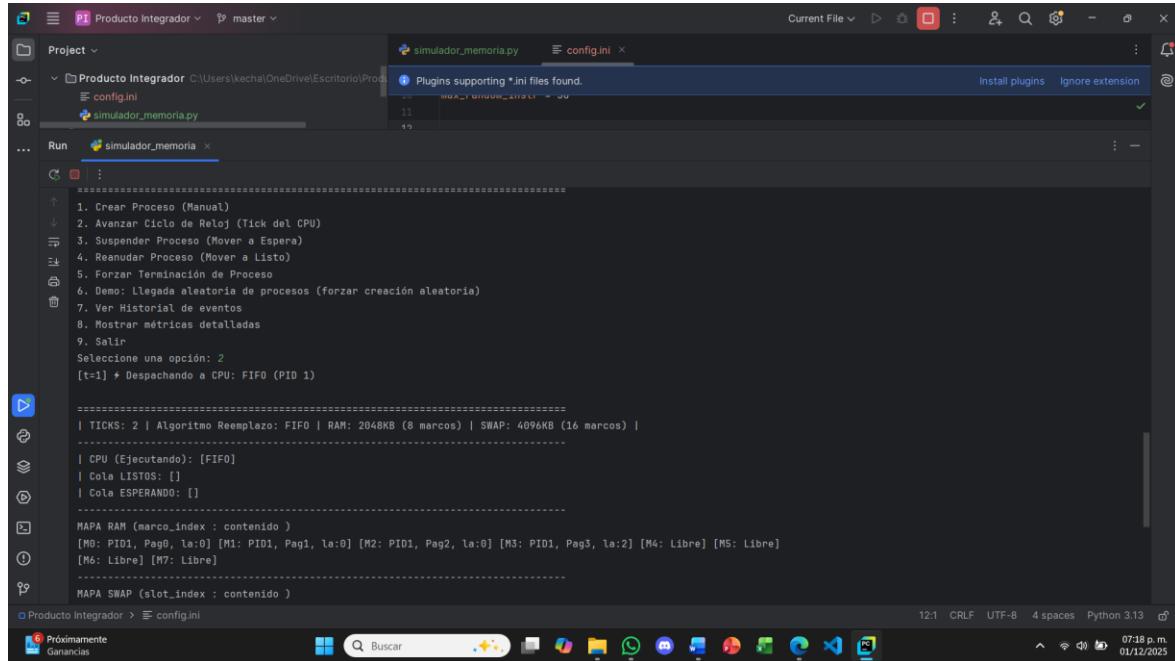
=====
| CPU ([Ejecutando]): [OCIOSO]
| Cola LISTOS: ['1:FIFO']
| Cola ESPERANDO: []
-----
MAPA RAM (marco_index : contenido)
[M0: PID1, Pag0, la:0] [M1: PID1, Pag1, la:0] [M2: PID1, Pag2, la:0] [M3: PID1, Pag3, la:0] [M4: Libre] [M5: Libre]
[M6: Libre] [M7: Libre]
-----
MAPA SWAP (slot_index : contenido)
[S0: Libre] [S1: Libre] [S2: Libre] [S3: Libre] [S4: Libre] [S5: Libre]
[S6: Libre] [S7: Libre] [S8: Libre] [S9: Libre] [S10: Libre] [S11: Libre]
[S12: Libre] [S13: Libre] [S14: Libre] [S15: Libre]
-----
TABLAS DE PÁGINAS (por proceso)
PID 1 FIFO: 0->M0, 1->M1, 2->M2, 3->M3
-----
METRICAS: Accesos=0, Fallos=0, SwapIns=0, SwapOuts=0, Util_RAM=4/8 (50.0%)
=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos

11:49 CRLF UTF-8 4 spaces Python 3.13
23°C Mayorm. nublado Buscar 07:14 p.m. 01/12/2025

```

Ciclo de Reloj y Ejecución

Aquí se visualiza la transición del sistema al avanzar un ciclo de reloj mediante la Opción 2. El planificador de procesos (Scheduler) ha seleccionado al proceso "FIFO" de la cola de listos y lo ha despachado al procesador, cambiando el estado del CPU a [FIFO]. El contador global de tiempo avanza (TICKS: 2) y se actualizan los metadatos de acceso en los marcos de memoria, lo cual es fundamental para alimentar la heurística de los algoritmos de reemplazo en ciclos futuros.



```
PT Producto Integrador master
Project
  -> Producto Integrador C:\Users\kecha\OneDrive\Escritorio\Producto Integrador
    config.ini
    simulador_memoria.py
Run simulador_memoria
:
=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos
8. Mostrar métricas detalladas
9. Salir
Seleccione una opción: 2
[t=1] * Despachando a CPU: FIFO (PID 1)

=====
| TICKS: 2 | Algoritmo Reemplazo: FIFO | RAM: 2048KB (8 marcos) | SWAP: 4096KB (16 marcos) |
-----
| CPU (Ejecutando): [FIFO]
| Cola LISTOS: []
| Cola ESPERANDO: []
-----
MAPA RAM (marco_index : contenido)
[M0: PID0, Pag0, la:0] [M1: PID1, Pag1, la:0] [M2: PID1, Pag2, la:0] [M3: PID1, Pag3, la:2] [M4: Libre] [M5: Libre]
[M6: Libre] [M7: Libre]
-----
MAPA SWAP (slot_index : contenido)

12:1 CRLF UTF-8 4 spaces Python 3.13
Próximamente Ganancias 07:18 p. m. 01/12/2025
```

Suspensión de Proceso

Esta captura documenta la gestión de estados del ciclo de vida de los procesos mediante la Opción 3. El usuario ha solicitado suspender el proceso activo (PID 1), lo que provoca una interrupción inmediata en su ejecución. El planificador responde moviendo el proceso de la cola de ejecución a la Cola ESPERANDO (Bloqueado), lo cual se verifica visualmente en la interfaz. Como consecuencia directa, el recurso del procesador se libera, quedando el estado del CPU como [OCIOSO], mientras el proceso retiene sus recursos de memoria pero detiene su consumo de ciclos de reloj.

```

MAPA RAM (marco_index : contenido )
[M0: PID1, Pag0, le:0] [M1: PID1, Pag1, le:0] [M2: PID1, Pag2, le:0] [M3: PID1, Pag3, le:2] [M4: Libre] [M5: Libre]
[M6: Libre] [M7: Libre]
-----
MAPA SWAP (slot_index : contenido )
[S0: Libre] [S1: Libre] [S2: Libre] [S3: Libre] [S4: Libre] [S5: Libre]
[S6: Libre] [S7: Libre] [S8: Libre] [S9: Libre] [S10: Libre] [S11: Libre]
[S12: Libre] [S13: Libre] [S14: Libre] [S15: Libre]
-----
TABLAS DE PAGINAS (por proceso)
PID 1 FIFO: 0->M0, 1->M1, 2->M2, 3->M3
-----
METRICAS: Accesos=1, Fallos=0, SwapIns=0, SwapOuts=0, Util_RAM=4/8 (50.0%)
Tasa de fallos: 0.00%
=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos
8. Mostrar métricas detalladas
9. Salir
Seleccione una opción:

```

Reanudación de Proceso

Aquí se ilustra la operación inversa utilizando la Opción 4 para desbloquear un proceso suspendido. Al ingresar el PID 1, el sistema emite el evento "Reanudado proceso FIFO", lo que indica una transición de estado de "Bloqueado" a "Listo". La evidencia técnica reside en la actualización de las colas: la lista de espera queda vacía y el proceso reaparece en la Cola LISTOS, quedando disponible nuevamente para ser elegible por el algoritmo de planificación en el siguiente tick del sistema.

```

=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos
8. Mostrar métricas detalladas
9. Salir
Seleccione una opción: 4
Ingrese PID a suspender: 1
[t=2] S Suspenido proceso en ejecución: FIFO

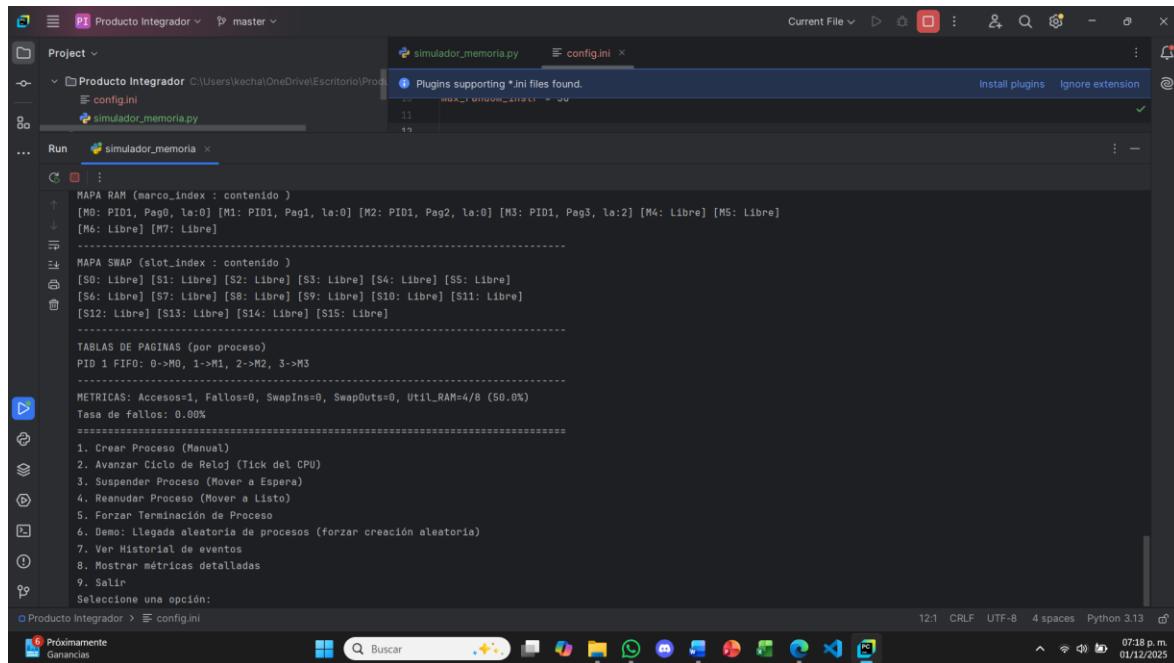
=====
| TICKS: 2 | Algoritmo Reemplazo: FIFO | RAM: 2048KB (8 marcos) | SWAP: 4096KB (16 marcos) |
| CPU (Ejecutando): [OCIOSO]
| Cola LISTOS: []
| Cola ESPERANDO: ['1:FIFO']

MAPA RAM (marco_index : contenido )
[M0: PID1, Pag0, le:0] [M1: PID1, Pag1, le:0] [M2: PID1, Pag2, le:0] [M3: PID1, Pag3, le:2] [M4: Libre] [M5: Libre]
[M6: Libre] [M7: Libre]
-----

```

Terminación Forzada

Esta imagen valida el mecanismo de recolección de basura (Garbage Collection) simulado a través de la Opción 5. Al forzar la terminación del proceso con PID 1, el sistema operativo realiza una limpieza integral de recursos. La sección crítica a observar es el Mapa RAM: los marcos M0, M1, M2 y M3, que previamente alojaban las páginas del proceso, han pasado instantáneamente al estado [Libre]. Esto confirma que el gestor de memoria ha invalidado correctamente las entradas en las tablas de páginas y ha recuperado el espacio físico para futuros procesos.



The screenshot shows a terminal window titled 'simulador_memoria.py' running in a project named 'Producto Integrador'. The terminal output displays memory maps and swap statistics. Key parts of the output include:

```
MAPA RAM (marco_index : contenido )
[M0: P101, Pag0, le:0] [M1: P101, Pag1, le:0] [M2: P101, Pag2, le:0] [M3: P101, Pag3, le:2] [M4: Libre] [M5: Libre]
[M6: Libre] [M7: Libre]

MAPA SWAP (slot_index : contenido )
[S0: Libre] [S1: Libre] [S2: Libre] [S3: Libre] [S4: Libre] [S5: Libre]
[S6: Libre] [S7: Libre] [S8: Libre] [S9: Libre] [S10: Libre] [S11: Libre]
[S12: Libre] [S13: Libre] [S14: Libre] [S15: Libre]

TABLAS DE PAGINAS (por proceso)
PID 1 FIFO: 0->M0, 1->M1, 2->M2, 3->M3

METRICAS: Accesos=1, Fallos=0, SwapIns=0, SwapOuts=0, Util_RAM=4/8 (50.0%)
Tasa de fallos: 0.00%
=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos
8. Mostrar métricas detalladas
9. Salir
Seleccione una opción:
```

Generación Aleatoria – Demo

En esta captura se utiliza la Opción 6 para probar la capacidad del sistema de manejar cargas de trabajo dinámicas. El modo "Demo" instancia automáticamente un proceso (PID 2, "P2_rand") con parámetros estocásticos: 475 KB de memoria y 2 páginas lógicas. El log de eventos muestra cómo el asignador de memoria busca los siguientes marcos disponibles, colocando las páginas 0 y 1 en los marcos físicos M4 y M5. Esto demuestra la estrategia de asignación del sistema, que utiliza los espacios libres de manera secuencial o basada en listas de huecos, coexistiendo con el estado actual de la memoria.

```

=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos
8. Mostrar métricas detalladas
9. Salir
Seleccione una opción: 4
Ingrese PID a reanudar: 1
[t=2] Reanudado proceso FIFO

=====
| TICKS: 2 | Algoritmo Reemplazo: FIFO | RAM: 2048KB (8 marcos) | SWAP: 4096KB (16 marcos) |

| CPU (Ejecutando): [0:IOS]
| Cola LISTOS: ['1:FIFO']
| Cola ESPERANDO: []

-----
MAPA RAM (marco_index : contenido )
[M0: PID1, Pag0, le:0] [M1: PID1, Pag1, le:0] [M2: PID1, Pag2, le:0] [M3: PID1, Pag3, le:2] [M4: Libre] [M5: Libre]
[M6: Libre] [M7: Libre]

-----
MAPA SWAP (slot_index : contenido )
[S0: Libre] [S1: Libre] [S2: Libre] [S3: Libre] [S4: Libre] [S5: Libre]
[S6: Libre] [S7: Libre] [S8: Libre] [S9: Libre] [S10: Libre] [S11: Libre]
[S12: Libre] [S13: Libre] [S14: Libre] [S15: Libre]

-----
TABLAS DE PAGINAS (por proceso)
PID 2 P2,rend: 0->M4, i->MS

-----
METRICAS: Accesos=1, Fallos=0, SwapIns=0, SwapOuts=0, Util_RAM=2/8 (25.0%)
Tasa de fallos: 0.00%
=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos
8. Mostrar métricas detalladas
9. Salir
Seleccione una opción: |

```

Histórico de Eventos

Esta captura documenta la capacidad de auditoría del simulador mediante la Opción 7, desplegando la traza cronológica completa de las operaciones realizadas por el Kernel simulado. Se valida la coherencia secuencial de las acciones: primero la creación, carga y ejecución del proceso PID 1, seguido de sus transiciones de estado (suspensión, reanudación) y su eventual terminación forzada, para finalizar con el registro de la llegada del proceso aleatorio PID 2, demostrando que el sistema mantiene un histórico persistente y ordenado de todas las transiciones y operaciones de memoria críticas.

```

=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos
8. Mostrar métricas detalladas
9. Salir
Seleccione una opción: 4
Ingrese PID a reanudar: 1
[t=2] Reanudado proceso FIFO

=====
| TICKS: 2 | Algoritmo Reemplazo: FIFO | RAM: 2048KB (8 marcos) | SWAP: 4096KB (16 marcos) |

| CPU (Ejecutando): [0:IOS]
| Cola LISTOS: ['1:FIFO']
| Cola ESPERANDO: []

-----
MAPA RAM (marco_index : contenido )
[M0: Libre] [M1: Libre] [M2: Libre] [M3: Libre] [M4: PID2, Pag0, le:2] [M5: PID2, Pag1, le:2]
[M6: Libre] [M7: Libre]

-----
MAPA SWAP (slot_index : contenido )
[S0: Libre] [S1: Libre] [S2: Libre] [S3: Libre] [S4: Libre] [S5: Libre]
[S6: Libre] [S7: Libre] [S8: Libre] [S9: Libre] [S10: Libre] [S11: Libre]
[S12: Libre] [S13: Libre] [S14: Libre] [S15: Libre]

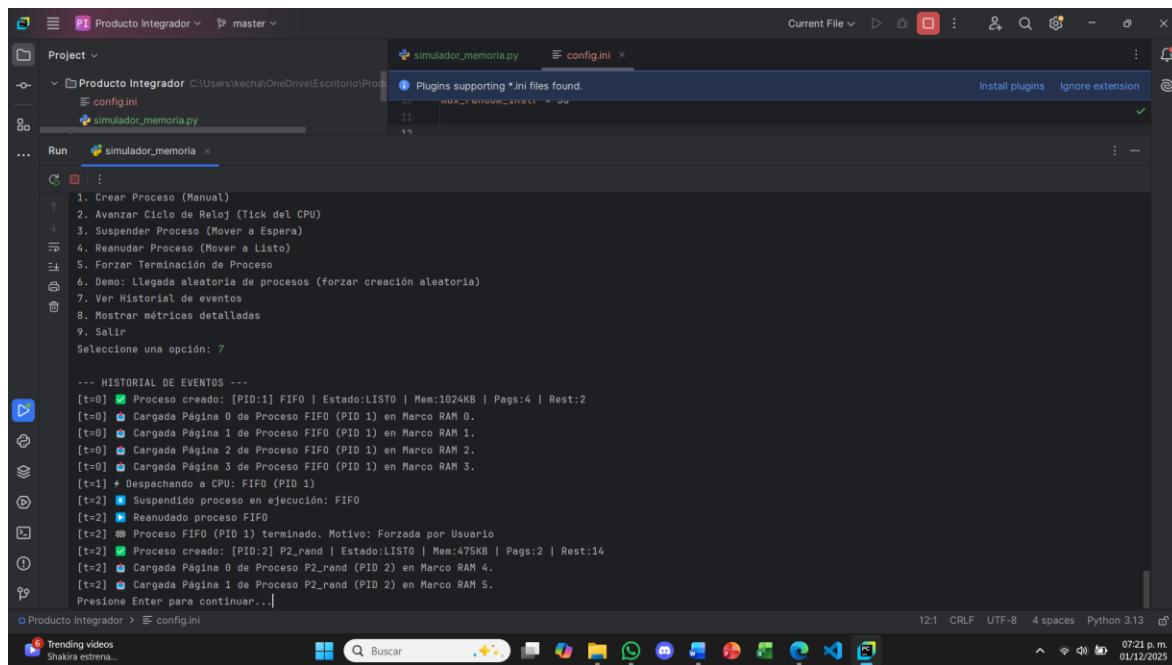
-----
TABLAS DE PAGINAS (por proceso)
PID 2 P2,rend: 0->M4, i->MS

-----
METRICAS: Accesos=1, Fallos=0, SwapIns=0, SwapOuts=0, Util_RAM=2/8 (25.0%)
Tasa de fallos: 0.00%
=====
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Historial de eventos
8. Mostrar métricas detalladas
9. Salir
Seleccione una opción: |

```

Métricas Detalladas

En esta imagen se visualiza el reporte estadístico de rendimiento generado por la Opción 8, reflejando el estado exacto del sistema tras las operaciones de limpieza. Los datos métricos indican una Utilización de RAM del 25.00% (2 marcos ocupados de 8 totales), lo cual es matemáticamente consistente con tener únicamente al proceso PID 2 activo en ese instante. Asimismo, la Tasa de Fallos se mantiene en 0.00%, confirmando que todos los accesos a memoria realizados hasta el ciclo de reloj actual (Tick 2) han sido aciertos en la memoria física, sin requerir acceso al disco.



The screenshot shows a terminal window with the following content:

```
PT Producto Integrador master
Project
  -> Producto Integrador C:\Users\kecha\OneDrive\Escritorio\Prod
    config.ini
    simulador_memoria.py
Run simulador_memoria
1. Crear Proceso (Manual)
2. Avanzar Ciclo de Reloj (Tick del CPU)
3. Suspender Proceso (Mover a Espera)
4. Reanudar Proceso (Mover a Listo)
5. Forzar Terminación de Proceso
6. Demo: Llegada aleatoria de procesos (forzar creación aleatoria)
7. Ver Histórial de eventos
8. Mostrar métricas detalladas
9. Salir
Seleccione una opción: 7

--- HISTÓRIAL DE EVENTOS ---
[t=0] ✅ Proceso creado: [PID:1] FIFO | Estado:LISTO | Mem:1024KB | Pags:4 | Rest:2
[t=0] 🚧 Cargada Página 0 de Proceso FIFO (PID 1) en Marco RAM 0.
[t=0] 🚧 Cargada Página 1 de Proceso FIFO (PID 1) en Marco RAM 1.
[t=0] 🚧 Cargada Página 2 de Proceso FIFO (PID 1) en Marco RAM 2.
[t=0] 🚧 Cargada Página 3 de Proceso FIFO (PID 1) en Marco RAM 3.
[t=1] ✅ Despachando a CPU: FIFO (PID 1)
[t=2] 🚧 Suspendido proceso en ejecución: FIFO
[t=2] ✅ Reanudado proceso FIFO
[t=2] ❌ Proceso FIFO (PID 1) terminado. Motivo: Forzada por Usuario
[t=2] ✅ Proceso creado: [PID:2] P2_rand | Estado:LISTO | Mem:475KB | Pags:2 | Rest:14
[t=2] 🚧 Cargada Página 0 de Proceso P2_rand (PID 2) en Marco RAM 4.
[t=2] 🚧 Cargada Página 1 de Proceso P2_rand (PID 2) en Marco RAM 5.

Presione Enter para continuar...|
```

Estado de Memoria y Fragmentación

Esta captura ofrece una evidencia visual técnica sobre la gestión de memoria y la recolección de basura (Garbage Collection). Se observa en el Mapa RAM que los marcos M0, M1, M2 y M3 han regresado al estado de "Libre", validando que el sistema recuperó correctamente y de inmediato el espacio físico del proceso terminado (PID 1). Simultáneamente, se aprecia el fenómeno de asignación dispersa, ya que el proceso activo PID 2 permanece asignado en los marcos superiores (M4 y M5), dejando un bloque de memoria disponible al inicio del espacio de direccionamiento físico, lo que ilustra cómo se generan huecos disponibles para futuros procesos.

```

| TICKS: 2 | Algoritmo Reemplazo: FIFO | RAM: 2048KB (8 marcos) | SWAP: 4096KB (16 marcos) |
| CPU ([Ejecutando]: [OCIOSO] | Cola LISTOS: ['2:P2_rand'] | Cola ESPERANDO: [] |
MAPA RAM (marco_index : contenido )
[M0: Libre] [M1: Libre] [M2: Libre] [M3: Libre] [M4: PID2, Pag0, la:2] [M5: PID2, Pag1, la:2]
[M6: Libre] [M7: Libre]

MAPA SWAP (slot_index : contenido )
[S0: Libre] [S1: Libre] [S2: Libre] [S3: Libre] [S4: Libre] [S5: Libre]
[S6: Libre] [S7: Libre] [S8: Libre] [S9: Libre] [S10: Libre] [S11: Libre]
[S12: Libre] [S13: Libre] [S14: Libre] [S15: Libre]

TABLAS DE PAGINAS (por proceso)
PID 2 P2_rand: 0->M4, 1->M5

METRICAS: Accesos=1, Fallos=0, SwapIns=0, SwapOuts=0, Util_RAM=2/8 (25.0%)
Tasa de fallos: 0.00%
=====
1. Crear Proceso (Manual)
```

Saturación de Memoria RAM

Esta captura es fundamental pues ilustra el escenario de agotamiento de recursos físicos. El usuario ingresa manualmente el "ProcesoB" (PID 2) de 1024 KB. Al requerir 4 páginas, el sistema las asigna en los únicos marcos restantes (M4 a M7), ya que los primeros (M0 a M3) estaban ocupados por el proceso FIFO. La evidencia visual crítica está en el Mapa RAM: no queda ningún marco marcado como [Libre]. La utilización de RAM ha alcanzado el 100% (8/8 marcos). Esto prepara el escenario perfecto para demostrar el Swapping, ya que cualquier solicitud de memoria futura obligará al sistema a expulsar una página al disco.

“Nota: Al llenarse la memoria RAM al 100% como se muestra en la imagen anterior, el sistema procede a ejecutar el algoritmo de reemplazo configurado (FIFO), moviendo las páginas víctima al área de Swap para liberar marcos, garantizando la continuidad operativa del sistema.”

```

| TICKS: 2 | Algoritmo Reemplazo: FIFO | RAM: 2048KB (8 marcos) | SWAP: 4096KB (16 marcos) |
| CPU ([Ejecutando]: [OCIOSO] | Cola LISTOS: ['2:P2_rand'] | Cola ESPERANDO: [] |
MAPA RAM (marco_index : contenido )
[M0: Libre] [M1: Libre] [M2: Libre] [M3: Libre] [M4: PID2, Pag0, la:2] [M5: PID2, Pag1, la:2]
[M6: Libre] [M7: Libre]

MAPA SWAP (slot_index : contenido )
[S0: Libre] [S1: Libre] [S2: Libre] [S3: Libre] [S4: Libre] [S5: Libre]
[S6: Libre] [S7: Libre] [S8: Libre] [S9: Libre] [S10: Libre] [S11: Libre]
[S12: Libre] [S13: Libre] [S14: Libre] [S15: Libre]

TABLAS DE PAGINAS (por proceso)
PID 2 P2_rand: 0->M4, 1->M5

METRICAS: Accesos=1, Fallos=0, SwapIns=0, SwapOuts=0, Util_RAM=2/8 (25.0%)
Tasa de fallos: 0.00%
=====
1. Crear Proceso (Manual)
```

Archivo de Configuración - config.ini

Esta captura documenta la interfaz de configuración persistente del sistema a través del archivo config.ini. Se visualizan los parámetros del hardware simulado definidos antes de la ejecución: una memoria RAM física de 2048 KB, un área de intercambio (Swap) de 4096 KB y un tamaño de página de 256 KB. Además, permite verificar las políticas operativas activas, como la selección del algoritmo de reemplazo "FIFO", la desactivación de la TLB (tlb_enabled = False) y una probabilidad de llegada de procesos estocásticos del 25% (arrival_prob = 0.25), lo cual establece la línea base sobre la que se realizaron todas las pruebas de rendimiento.

```
=====
| TICKS: 0 | Algoritmo Reemplazo: FIFO | RAM: 2048KB (8 marcos) | SWAP: 4096KB (16 marcos) |
-----
| CPU (Ejecutando): [OCIOSO]
| Cola LISTOS: ['1:ProcesoA', '2:ProcesoB']
| Cola ESPERANDO: []

MAPA RAM (marco_index : contenido )
[M0: PID1, Pag0, la:0] [M1: PID1, Pag1, la:0] [M2: PID1, Pag2, la:0] [M3: PID1, Pag3, la:0] [M4: PID2, Pag0, la:0] [M5: PID2, Pag1, la:0]
[M6: PID2, Pag2, la:0] [M7: PID2, Pag3, la:0]

MAPA SWAP (slot_index : contenido )
[S0: Libre] [S1: Libre] [S2: Libre] [S3: Libre] [S4: Libre] [S5: Libre]
[S6: Libre] [S7: Libre] [S8: Libre] [S9: Libre] [S10: Libre] [S11: Libre]
[S12: Libre] [S13: Libre] [S14: Libre] [S15: Libre]

TABLAS DE PAGINAS (por proceso)
PID 1 ProcesoA: 0->M0, 1->M1, 2->M2, 3->M3
PID 2 ProcesoB: 0->M4, 1->M5, 2->M6, 3->M7

METRICAS: Accesos=0, Fallos=0, SwapIns=0, SwapOuts=0, Util_RAM=8/8 (100.0%)
=====
```