1. Descripción General del Proyecto

El objetivo del proyecto es que los estudiantes desarrollen un **simulador funcional del gestor de memoria RAM y el área de intercambio (Swap)** de un sistema operativo, que permita visualizar cómo un SO asigna recursos, traduce direcciones y maneja situaciones de escasez de memoria en un entorno multiprogramado.

El resultado será una aplicación (consola, o GUI), acompañada de documentación técnica y alojada en un repositorio GitHub. El enlace al repositorio será entregado en Moodle como evidencia final.

El simulador recibirá como entrada una serie de "procesos" con requisitos de memoria específicos y deberá asignarlos utilizando un esquema de **Paginación**. El sistema deberá mostrar visualmente el estado de la memoria física (marcos) y lógica (páginas), y tomar decisiones de reemplazo cuando la memoria principal se llene.

2. Objetivos del Proyecto

Que el estudiante:

- Comprenda y modele el funcionamiento interno del gestor de memoria.
- Aplique conceptos teóricos de la unidad realizando un producto funcional.
- Produzca documentación técnica clara y estructurada.

3. Requerimientos Generales

1. El desarrollo debe estar alojado en un repositorio GitHub público o privado con acceso al profesor.
2. El repositorio debe contener:
    a. Código fuente limpio y comentado.
    b. Manual de usuario
    c. Manual técnico
    d. Evidencia de pruebas (capturas, casos, logs)
    e. Archivo README principal:
        i. Nombre de los integrantes.

        ii. Instrucciones claras de cómo compilar y ejecutar el simulador.
        iii. Explicación breve del diseño: qué estructuras de datos usaron para las tablas de páginas y la memoria, y qué algoritmo de reemplazo implementaron.
3. La entrega final será un enlace al repositorio dentro de Moodle y un reporte Técnico (PDF) incluyendo:
    a. Portada con datos del equipo.
    b. Descripción de la implementación de cada módulo.
    c. Ejemplos de ejecución con capturas de pantalla.
    d. Análisis comparativo de los algoritmos implementados. (si fueron varios distintos)

4. Requisitos Funcionales

El simulador debe cumplir indispensablemente con las siguientes funcionalidades:

A. Configuración inicial del sistema. Al iniciar, el simulador debe permitir configurar mediante archivo config.ini o similar:
    a. Tamaño total de la Memoria Física (RAM) simulada (ej. 2048 KB).
    b. Tamaño total del Área de Intercambio (Swap) simulada (ej. 4096 KB).
    c. Tamaño de página/marco (ej. 256 KB). Nota: Esto determinará cuántos marcos totales tiene su RAM y Swap.
B. Gestión de procesos en entorno de multiprogramación. El sistema debe permitir la "llegada" y "salida" de procesos dinámicamente (o sea al azar).
C. Motor de Paginación y Asignación. Al llegar un proceso, el simulador de manera general, debe:
    a. Calcular cuántas páginas necesita el proceso basado en su tamaño y el tamaño de página del sistema.
    b. Colocarlos en memoria siguiendo uno de los esquemas estudiados y actualizar la Tabla de Páginas/Segmentos del

proceso (y en su caso TLB) así como el mapa de bits (o bit de validación) de la memoria general.

c. Si no hay suficientes marcos libres en RAM, se debe activar el mecanismo de Swapping (ver punto E).

D. Visualización del Estado de la Memoria (Reportes) El usuario debe poder visualizar en cualquier momento el estado del sistema mediante la interfaz proporcionada CLI/GUI:
   a. Lista de procesos activos, suspendidos o intercambiados.
   b. Estado: Debe mostrar un "mapa" de la memoria RAM y del Swap, indicando qué proceso ocupa cada marco y qué marcos están libres.
      i. Representación gráfica o textual del mapa de memoria: Ejemplo visual simple: [Marco 0: ProcesoA, Pag 0] [Marco 1: Libre] [Marco 2: ProcesoB, Pag 1] ...
      ii. Representación de tablas de páginas : tabla <ID_proceso>: Debe mostrar la tabla de páginas específica de un proceso activo (Página Lógica X -> Marco Físico Y).
   c. Movimientos de memoria y registros de fallos de página

E. Swapping y Memoria Virtual (Algoritmos de Reemplazo) Si llega un proceso y la RAM está llena:
   a. Debe implementarse al menos un algoritmo de reemplazo de páginas visto en clase: FIFO (First-In, First-Out) o LRU (Least Recently Used), reloj. Para elegir una "página víctima" de la RAM para enviarla al Swap.
   b. El simulador debe mover la página víctima de RAM al Swap, actualizar las tablas correspondientes (marcando la página como "no presente" en RAM pero sí en disco), y usar el marco liberado para el nuevo proceso.
   c. Debe emitirse un mensaje indicando la operación de swapping: "Página X del Proceso Y movida a Swap por falta de espacio."

F. Métricas de rendimiento (tasa de fallos, utilización de memoria).

5. Requerimientos Técnicos

- Puede desarrollarse en el lenguaje de su elección, aunque se recomienda Rust, C o C++ por su cercanía al manejo de memoria, también puede desarrollarse en Java o Python por su facilidad para estructuras de datos.
- Debe ser completamente ejecutable desde cualquier equipo sin configuraciones especiales.
- Interfaz: Gráfica (GUI) o por terminal (CLI).

6. Estructura Mínima

```
/src        -> Código fuente
/docs       -> Manual técnico y manual del usuario
/tests      -> Pruebas y capturas
README.md   -> Documentación principal
LICENSE     -> (Opcional) Tipo de licencia
```

VERDAD, BELLEZA, PROBIDAD

UNIVERSIDAD AUTÓNOMA DE TAMAULIPAS

FACULTAD DE INGENIERÍA

Producto integrador final

Asignatura:

Sistemas Operativos

Integrantes:

- Solbes Davalos Rodrigo
- Izaguirre Cortes Emanuel
- Rosales Pereles Denisse Ariadna
- Morales Urrutia Javier Antonio
- Reyes Alejo Emiliano

SEMESTRE: 6° GRUPO: g

CARRERA:

ING.SISTEMAS COMPUTACIONALES

PROFESOR:

Muñoz Quintero Dante Adolfo

CICLO ESCOLAR

2025_3

Código

```
/*
 * Simulador de Gestor de Memoria RAM y Swap
 * Sistemas Operativos - Universidad Autónoma de Tamaulipas
 *
 * Autores:
 * - Solbes Davalos Rodrigo
 * - Izaguirre Cortes Emanuel
 * - Rosales Pereles Denisse Ariadna
 * - Morales Urrutia Javier Antonio
 * - Reyes Alejo Emiliano
 *
 * Profesor: Dante Adolfo Muñoz Quintero
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdbool.h>

// ==================== CONSTANTES Y CONFIGURACIÓN ====================

#define MAX_PROCESSES 50
```

VERDAD, BELLEZA, PROBIDAD

```
#define MAX_LOG_ENTRIES 1000

#define MAX_LINE_LENGTH 256


// Configuración por defecto (se puede sobrescribir con config.ini)

int RAM_SIZE = 2048;        // KB

int SWAP_SIZE = 4096;       // KB

int PAGE_SIZE = 256;        // KB

int TLB_SIZE = 4;           // Número de entradas en TLB


// Variables calculadas

int NUM_RAM_FRAMES;         // Número de marcos en RAM

int NUM_SWAP_FRAMES;        // Número de marcos en Swap


// ==================== ESTRUCTURAS DE DATOS ====================


// Estados de una página

typedef enum {

    PAGE_FREE,              // Página libre

    PAGE_IN_RAM,           // Página en RAM

    PAGE_IN_SWAP,          // Página en Swap

    PAGE_NOT_PRESENT       // Página no presente

} PageState;


// Estados de un proceso

typedef enum {
```

VERDAD, BELLEZA, PROBIDAD

```
    PROC_ACTIVE,            // Proceso activo
    PROC_SUSPENDED,         // Proceso suspendido
    PROC_SWAPPED,           // Proceso intercambiado (algunas páginas en swap)
    PROC_TERMINATED         // Proceso terminado
} ProcessState;

// Entrada de la tabla de páginas
typedef struct {
    int page_number;        // Número de página lógica
    int frame_number;       // Número de marco físico (-1 si no está en RAM)
    PageState state;        // Estado de la página
    bool valid;             // Bit de validez
    bool modified;          // Bit de modificación (dirty bit)
    int swap_position;      // Posición en swap (-1 si no está en swap)
    time_t last_access;     // Timestamp del último acceso (para LRU)
    time_t load_time;       // Timestamp de carga (para FIFO)
} PageTableEntry;

// Bloque de Control de Proceso (PCB)
typedef struct {
    int pid;                    // ID del proceso
    char name[32];              // Nombre del proceso
    int size;                   // Tamaño en KB
    int num_pages;              // Número de páginas necesarias
    ProcessState state;         // Estado del proceso
```

VERDAD, BELLEZA, PROBIDAD

```
    PageTableEntry *page_table;     // Tabla de páginas
    time_t creation_time;           // Tiempo de creación
    int page_faults;                // Contador de fallos de página
} PCB;

// Entrada de la TLB (Translation Lookaside Buffer)
typedef struct {
    int pid;                // PID del proceso
    int page_number;        // Número de página
    int frame_number;       // Número de marco
    bool valid;             // Entrada válida
    time_t last_access;     // Para reemplazo en TLB
} TLBEntry;

// Marco de memoria (RAM o Swap)
typedef struct {
    int pid;                // PID del proceso que ocupa el marco (-1 si libre)
    int page_number;        // Número de página del proceso
    bool occupied;          // Marco ocupado
    time_t load_time;       // Tiempo de carga (para FIFO)
} Frame;

// Cola FIFO para algoritmo de reemplazo
typedef struct {
    int *queue;             // Array de índices de marcos
```

VERDAD, BELLEZA, PROBIDAD

```
    int front;              // Frente de la cola
    int rear;               // Final de la cola
    int size;               // Tamaño actual
    int capacity;           // Capacidad máxima
} FIFOQueue;

// Entrada de log
typedef struct {
    time_t timestamp;
    char message[256];
} LogEntry;

// Sistema de memoria
typedef struct {
    Frame *ram_frames;              // Marcos de RAM
    Frame *swap_frames;             // Marcos de Swap
    PCB *processes[MAX_PROCESSES];  // Procesos activos
    int num_processes;              // Número de procesos activos
    TLBEntry *tlb;                  // TLB
    FIFOQueue *fifo_queue;          // Cola FIFO para reemplazo
    LogEntry *logs;                 // Sistema de logs
    int log_count;                  // Contador de logs
    int total_page_faults;          // Total de fallos de página
    int total_swaps;                // Total de operaciones de swap
    int total_tlb_hits;             // Total de aciertos en TLB
```

```
    int total_tlb_misses;           // Total de fallos en TLB
    int total_memory_accesses;      // Total de accesos a memoria
    time_t start_time;              // Tiempo de inicio
} MemorySystem;


// Variable global del sistema
MemorySystem *mem_system = NULL;


// ==================== PROTOTIPOS DE FUNCIONES ====================


// Inicialización y configuración
void load_config(const char *filename);
void init_system();
void free_system();


// Cola FIFO
FIFOQueue* create_fifo_queue(int capacity);
void enqueue_fifo(FIFOQueue *queue, int frame_index);
int dequeue_fifo(FIFOQueue *queue);
bool is_fifo_empty(FIFOQueue *queue);
void free_fifo_queue(FIFOQueue *queue);


// Gestión de procesos
int create_process(const char *name, int size_kb);
bool terminate_process(int pid);
```

VERDAD, BELLEZA, PROBIDAD

```
PCB* find_process(int pid);


// Gestión de memoria
int allocate_page_in_ram(int pid, int page_number);
int allocate_page_in_swap(int pid, int page_number);
int find_free_ram_frame();
int find_free_swap_frame();
int select_victim_page_fifo();
bool swap_out_page(int frame_index);
bool swap_in_page(int pid, int page_number);


// TLB
void init_tlb();
int tlb_lookup(int pid, int page_number);
void tlb_update(int pid, int page_number, int frame_number);
void tlb_invalidate(int pid);


// Visualización
void display_memory_map();
void display_process_table(int pid);
void display_system_status();
void display_statistics();
void display_tlb();


// Logs
```

VERDAD, BELLEZA, PROBIDAD

```
void add_log(const char *message);

void display_logs(int count);

void save_logs_to_file(const char *filename);


// Menú y utilidades

void print_menu();

void run_simulator();

int get_user_input_int(const char *prompt);

void clear_screen();

void pause_screen();


// ==================== IMPLEMENTACIÓN DE FUNCIONES ====================


// Cargar configuración desde archivo

void load_config(const char *filename) {

    FILE *file = fopen(filename, "r");

    if (!file) {

        printf("⚠️  Archivo de configuración no encontrado. Usando valores por
defecto.\n");

        return;

    }


    char line[MAX_LINE_LENGTH];

    while (fgets(line, sizeof(line), file)) {

        // Ignorar comentarios y líneas vacías

        if (line[0] == '#' || line[0] == '\n' || line[0] == '[') continue;
```

VERDAD, BELLEZA, PROBIDAD

```
            char key[64], value[64];
            if (sscanf(line, "%s = %s", key, value) == 2) {
                if (strcmp(key, "RAM_SIZE") == 0) {
                    RAM_SIZE = atoi(value);
                } else if (strcmp(key, "SWAP_SIZE") == 0) {
                    SWAP_SIZE = atoi(value);
                } else if (strcmp(key, "PAGE_SIZE") == 0) {
                    PAGE_SIZE = atoi(value);
                } else if (strcmp(key, "TLB_SIZE") == 0) {
                    TLB_SIZE = atoi(value);
                }
            }
        }

    fclose(file);
    printf("✓ Configuración cargada desde %s\n", filename);
}

// Crear cola FIFO
FIFOQueue* create_fifo_queue(int capacity) {
    FIFOQueue *queue = (FIFOQueue*)malloc(sizeof(FIFOQueue));
    queue->queue = (int*)malloc(capacity * sizeof(int));
    queue->front = 0;
    queue->rear = -1;
```

VERDAD, BELLEZA, PROBIDAD

```
    queue->size = 0;

    queue->capacity = capacity;

    return queue;

}


// Agregar a cola FIFO

void enqueue_fifo(FIFOQueue *queue, int frame_index) {

    if (queue->size >= queue->capacity) {

        return; // Cola llena

    }

    queue->rear = (queue->rear + 1) % queue->capacity;

    queue->queue[queue->rear] = frame_index;

    queue->size++;

}


// Remover de cola FIFO

int dequeue_fifo(FIFOQueue *queue) {

    if (queue->size == 0) {

        return -1; // Cola vacía

    }

    int frame_index = queue->queue[queue->front];

    queue->front = (queue->front + 1) % queue->capacity;

    queue->size--;

    return frame_index;

}
```

VERDAD, BELLEZA, PROBIDAD

```
// Verificar si cola está vacía
bool is_fifo_empty(FIFOQueue *queue) {
    return queue->size == 0;
}

// Liberar cola FIFO
void free_fifo_queue(FIFOQueue *queue) {
    free(queue->queue);
    free(queue);
}

// Inicializar TLB
void init_tlb() {
    mem_system->tlb = (TLBEntry*)malloc(TLB_SIZE * sizeof(TLBEntry));
    for (int i = 0; i < TLB_SIZE; i++) {
        mem_system->tlb[i].valid = false;
        mem_system->tlb[i].pid = -1;
        mem_system->tlb[i].page_number = -1;
        mem_system->tlb[i].frame_number = -1;
        mem_system->tlb[i].last_access = 0;
    }
}

// Buscar en TLB
```

VERDAD, BELLEZA, PROBIDAD

```
int tlb_lookup(int pid, int page_number) {
    for (int i = 0; i < TLB_SIZE; i++) {
        if (mem_system->tlb[i].valid &&
            mem_system->tlb[i].pid == pid &&
            mem_system->tlb[i].page_number == page_number) {
            mem_system->tlb[i].last_access = time(NULL);
            mem_system->total_tlb_hits++;
            return mem_system->tlb[i].frame_number;
        }
    }
    mem_system->total_tlb_misses++;
    return -1; // TLB miss
}

// Actualizar TLB
void tlb_update(int pid, int page_number, int frame_number) {
    // Buscar entrada vacía o la más antigua
    int victim_index = 0;
    time_t oldest_time = mem_system->tlb[0].last_access;

    for (int i = 0; i < TLB_SIZE; i++) {
        if (!mem_system->tlb[i].valid) {
            victim_index = i;
            break;
        }
```

VERDAD, BELLEZA, PROBIDAD

```
        if (mem_system->tlb[i].last_access < oldest_time) {
            oldest_time = mem_system->tlb[i].last_access;
            victim_index = i;
        }
    }

    // Actualizar entrada
    mem_system->tlb[victim_index].pid = pid;
    mem_system->tlb[victim_index].page_number = page_number;
    mem_system->tlb[victim_index].frame_number = frame_number;
    mem_system->tlb[victim_index].valid = true;
    mem_system->tlb[victim_index].last_access = time(NULL);
}

// Invalidar entradas de TLB de un proceso
void tlb_invalidate(int pid) {
    for (int i = 0; i < TLB_SIZE; i++) {
        if (mem_system->tlb[i].valid && mem_system->tlb[i].pid == pid) {
            mem_system->tlb[i].valid = false;
        }
    }
}

// Inicializar sistema de memoria
void init_system() {
```

VERDAD, BELLEZA, PROBIDAD

```
    // Calcular número de marcos
    NUM_RAM_FRAMES = RAM_SIZE / PAGE_SIZE;
    NUM_SWAP_FRAMES = SWAP_SIZE / PAGE_SIZE;


printf("\n╔═══════════════════════════════════════════════════════════╗\n");
    printf("║     INICIALIZANDO SIMULADOR DE GESTOR DE MEMORIA          ║\n");
printf("╚═══════════════════════════════════════════════════════════╝\n\n");

    // Crear estructura del sistema
    mem_system = (MemorySystem*)malloc(sizeof(MemorySystem));

    // Inicializar marcos de RAM
    mem_system->ram_frames = (Frame*)malloc(NUM_RAM_FRAMES * sizeof(Frame));
    for (int i = 0; i < NUM_RAM_FRAMES; i++) {
        mem_system->ram_frames[i].pid = -1;
        mem_system->ram_frames[i].page_number = -1;
        mem_system->ram_frames[i].occupied = false;
        mem_system->ram_frames[i].load_time = 0;
    }

    // Inicializar marcos de Swap
    mem_system->swap_frames = (Frame*)malloc(NUM_SWAP_FRAMES * sizeof(Frame));
    for (int i = 0; i < NUM_SWAP_FRAMES; i++) {
        mem_system->swap_frames[i].pid = -1;
```

VERDAD, BELLEZA, PROBIDAD

```
        mem_system->swap_frames[i].page_number = -1;
        mem_system->swap_frames[i].occupied = false;
        mem_system->swap_frames[i].load_time = 0;
    }

    // Inicializar procesos
    mem_system->num_processes = 0;
    for (int i = 0; i < MAX_PROCESSES; i++) {
        mem_system->processes[i] = NULL;
    }

    // Inicializar TLB
    init_tlb();

    // Inicializar cola FIFO
    mem_system->fifo_queue = create_fifo_queue(NUM_RAM_FRAMES);

    // Inicializar logs
    mem_system->logs = (LogEntry*)malloc(MAX_LOG_ENTRIES * sizeof(LogEntry));
    mem_system->log_count = 0;

    // Inicializar estadísticas
    mem_system->total_page_faults = 0;
    mem_system->total_swaps = 0;
    mem_system->total_tlb_hits = 0;
```

VERDAD, BELLEZA, PROBIDAD

```
    mem_system->total_tlb_misses = 0;
    mem_system->total_memory_accesses = 0;
    mem_system->start_time = time(NULL);

    printf("✓ Memoria RAM inicializada: %d KB (%d marcos de %d KB)\n",
           RAM_SIZE, NUM_RAM_FRAMES, PAGE_SIZE);
    printf("✓ Área de Swap inicializada: %d KB (%d marcos de %d KB)\n",
           SWAP_SIZE, NUM_SWAP_FRAMES, PAGE_SIZE);
    printf("✓ TLB inicializada: %d entradas\n", TLB_SIZE);
    printf("✓ Algoritmo de reemplazo: FIFO\n");

    add_log("Sistema de memoria inicializado correctamente");
}

// Liberar sistema
void free_system() {
    if (!mem_system) return;

    // Liberar procesos
    for (int i = 0; i < MAX_PROCESSES; i++) {
        if (mem_system->processes[i]) {
            free(mem_system->processes[i]->page_table);
            free(mem_system->processes[i]);
        }
    }
```

```
    // Liberar estructuras
    free(mem_system->ram_frames);
    free(mem_system->swap_frames);
    free(mem_system->tlb);
    free_fifo_queue(mem_system->fifo_queue);
    free(mem_system->logs);
    free(mem_system);

    printf("\n✓ Sistema liberado correctamente.\n");
}

// Agregar entrada al log
void add_log(const char *message) {
    if (mem_system->log_count >= MAX_LOG_ENTRIES) {
        return; // Log lleno
    }

    LogEntry *entry = &mem_system->logs[mem_system->log_count++];
    entry->timestamp = time(NULL);
    strncpy(entry->message, message, sizeof(entry->message) - 1);
    entry->message[sizeof(entry->message) - 1] = '\0';
}

// Buscar proceso por PID
```

VERDAD, BELLEZA, PROBIDAD

```
PCB* find_process(int pid) {
    for (int i = 0; i < MAX_PROCESSES; i++) {
        if (mem_system->processes[i] && mem_system->processes[i]->pid == pid) {
            return mem_system->processes[i];
        }
    }
    return NULL;
}

// Buscar marco libre en RAM
int find_free_ram_frame() {
    for (int i = 0; i < NUM_RAM_FRAMES; i++) {
        if (!mem_system->ram_frames[i].occupied) {
            return i;
        }
    }
    return -1; // No hay marcos libres
}

// Buscar marco libre en Swap
int find_free_swap_frame() {
    for (int i = 0; i < NUM_SWAP_FRAMES; i++) {
        if (!mem_system->swap_frames[i].occupied) {
            return i;
        }
```

VERDAD, BELLEZA, PROBIDAD

```
    }

    return -1; // No hay marcos libres

}


// Seleccionar página víctima usando FIFO

int select_victim_page_fifo() {

    if (is_fifo_empty(mem_system->fifo_queue)) {

        return -1;

    }

    int victim = dequeue_fifo(mem_system->fifo_queue);

    return victim;

}


// Intercambiar página de RAM a Swap (Swap Out)

bool swap_out_page(int frame_index) {

    if (frame_index < 0 || frame_index >= NUM_RAM_FRAMES) {

        return false;

    }


    Frame *ram_frame = &mem_system->ram_frames[frame_index];

    if (!ram_frame->occupied) {

        return false;

    }


    // Buscar proceso y página
```

VERDAD, BELLEZA, PROBIDAD

```
    PCB *process = find_process(ram_frame->pid);
    if (!process) {
        return false;
    }

    int page_number = ram_frame->page_number;
    PageTableEntry *page_entry = &process->page_table[page_number];

    // Buscar espacio en Swap
    int swap_frame = find_free_swap_frame();
    if (swap_frame == -1) {
        char msg[256];
        snprintf(msg, sizeof(msg), "ERROR: No hay espacio en Swap para el Proceso
%d, Página %d",
                 ram_frame->pid, page_number);
        add_log(msg);
        return false;
    }

    // Mover página a Swap
    mem_system->swap_frames[swap_frame].pid = ram_frame->pid;
    mem_system->swap_frames[swap_frame].page_number = page_number;
    mem_system->swap_frames[swap_frame].occupied = true;
    mem_system->swap_frames[swap_frame].load_time = time(NULL);

    // Actualizar tabla de páginas
```

VERDAD, BELLEZA, PROBIDAD

```
    page_entry->state = PAGE_IN_SWAP;
    page_entry->frame_number = -1;
    page_entry->swap_position = swap_frame;
    page_entry->valid = false;

    // Liberar marco en RAM
    ram_frame->pid = -1;
    ram_frame->page_number = -1;
    ram_frame->occupied = false;

    // Invalidar entrada en TLB
    tlb_invalidate(process->pid);

    // Actualizar estadísticas
    mem_system->total_swaps++;

    char msg[256];
    snprintf(msg, sizeof(msg), "SWAP OUT: Proceso %d, Página %d movida de RAM[%d]
a Swap[%d]",
             process->pid, page_number, frame_index, swap_frame);
    add_log(msg);

    return true;
}

// Intercambiar página de Swap a RAM (Swap In)
```

VERDAD, BELLEZA, PROBIDAD

```
bool swap_in_page(int pid, int page_number) {
    PCB *process = find_process(pid);
    if (!process) {
        return false;
    }

    if (page_number < 0 || page_number >= process->num_pages) {
        return false;
    }

    PageTableEntry *page_entry = &process->page_table[page_number];

    if (page_entry->state != PAGE_IN_SWAP) {
        return false;
    }

    int swap_position = page_entry->swap_position;
    if (swap_position < 0 || swap_position >= NUM_SWAP_FRAMES) {
        return false;
    }

    // Buscar marco libre en RAM
    int ram_frame = find_free_ram_frame();

    // Si no hay marcos libres, hacer swap out de una página
```

```
    if (ram_frame == -1) {
        int victim_frame = select_victim_page_fifo();
        if (victim_frame == -1) {
            char msg[256];
            snprintf(msg, sizeof(msg), "ERROR: No se pudo encontrar página víctima
para Proceso %d", pid);
            add_log(msg);
            return false;
        }

        if (!swap_out_page(victim_frame)) {
            return false;
        }

        ram_frame = victim_frame;
    }

    // Mover página de Swap a RAM
    mem_system->ram_frames[ram_frame].pid = pid;
    mem_system->ram_frames[ram_frame].page_number = page_number;
    mem_system->ram_frames[ram_frame].occupied = true;
    mem_system->ram_frames[ram_frame].load_time = time(NULL);

    // Actualizar tabla de páginas
    page_entry->state = PAGE_IN_RAM;
    page_entry->frame_number = ram_frame;
```

VERDAD, BELLEZA, PROBIDAD

```
    page_entry->swap_position = -1;
    page_entry->valid = true;
    page_entry->load_time = time(NULL);

    // Liberar marco en Swap
    mem_system->swap_frames[swap_position].pid = -1;
    mem_system->swap_frames[swap_position].page_number = -1;
    mem_system->swap_frames[swap_position].occupied = false;

    // Actualizar TLB
    tlb_update(pid, page_number, ram_frame);

    // Agregar a cola FIFO
    enqueue_fifo(mem_system->fifo_queue, ram_frame);

    // Actualizar estadísticas
    mem_system->total_swaps++;
    process->page_faults++;
    mem_system->total_page_faults++;

    char msg[256];
    snprintf(msg, sizeof(msg), "SWAP IN: Proceso %d, Página %d movida de Swap[%d]
a RAM[%d]",
             pid, page_number, swap_position, ram_frame);
    add_log(msg);
```

VERDAD, BELLEZA, PROBIDAD

```
    return true;
}

// Asignar página en RAM
int allocate_page_in_ram(int pid, int page_number) {
    // Buscar marco libre
    int frame_index = find_free_ram_frame();

    // Si no hay marcos libres, hacer swap out
    if (frame_index == -1) {
        int victim_frame = select_victim_page_fifo();
        if (victim_frame == -1) {
            return -1; // No se pudo hacer swap out
        }

        if (!swap_out_page(victim_frame)) {
            return -1;
        }

        frame_index = victim_frame;
    }

    // Asignar marco
    mem_system->ram_frames[frame_index].pid = pid;
    mem_system->ram_frames[frame_index].page_number = page_number;
```

VERDAD, BELLEZA, PROBIDAD

```
    mem_system->ram_frames[frame_index].occupied = true;
    mem_system->ram_frames[frame_index].load_time = time(NULL);

    // Agregar a cola FIFO
    enqueue_fifo(mem_system->fifo_queue, frame_index);

    return frame_index;
}

// Crear proceso - CORREGIDO PARA NO HACER SWAP OUT AL CREAR
int create_process(const char *name, int size_kb) {
    if (mem_system->num_processes >= MAX_PROCESSES) {
        printf("❌ Error: Número máximo de procesos alcanzado.\n");
        return -1;
    }

    if (size_kb <= 0) {
        printf("❌ Error: Tamaño de proceso inválido.\n");
        return -1;
    }

    // Calcular número de páginas necesarias
    int num_pages = (size_kb + PAGE_SIZE - 1) / PAGE_SIZE; // Redondeo hacia
arriba

    // Verificar si hay suficiente espacio (RAM + Swap)
```

```
int total_space = (NUM_RAM_FRAMES + NUM_SWAP_FRAMES) * PAGE_SIZE;

int used_space = 0;


for (int i = 0; i < NUM_RAM_FRAMES; i++) {

    if (mem_system->ram_frames[i].occupied) {

        used_space += PAGE_SIZE;

    }

}

for (int i = 0; i < NUM_SWAP_FRAMES; i++) {

    if (mem_system->swap_frames[i].occupied) {

        used_space += PAGE_SIZE;

    }

}


if (used_space + (num_pages * PAGE_SIZE) > total_space) {

    printf("❌ Error: No hay suficiente espacio en memoria (RAM + Swap).\n");

    return -1;

}


// Buscar slot libre para el proceso

int slot = -1;

for (int i = 0; i < MAX_PROCESSES; i++) {

    if (mem_system->processes[i] == NULL) {

        slot = i;

        break;
```

VERDAD, BELLEZA, PROBIDAD

```
        }
    }

    if (slot == -1) {
        printf("❌ Error: No hay slots disponibles para procesos.\n");
        return -1;
    }

    // Crear PCB
    PCB *process = (PCB*)malloc(sizeof(PCB));
    process->pid = slot + 1; // PID comienza en 1
    strncpy(process->name, name, sizeof(process->name) - 1);
    process->name[sizeof(process->name) - 1] = '\0';
    process->size = size_kb;
    process->num_pages = num_pages;
    process->state = PROC_ACTIVE;
    process->creation_time = time(NULL);
    process->page_faults = 0;

    // Crear tabla de páginas
    process->page_table = (PageTableEntry*)malloc(num_pages * sizeof(PageTableEntry));

    // LÓGICA CORREGIDA: Asignar páginas SOLO EN MARCOS LIBRES
    // No hacer swap out de procesos existentes al crear uno nuevo
    int pages_in_ram = 0;
```

VERDAD, BELLEZA, PROBIDAD

```
    for (int i = 0; i < num_pages; i++) {
        // Verificar si hay marco libre en RAM
        int frame = find_free_ram_frame();

        if (frame != -1) {
            // Asignar directamente en RAM (sin hacer swap out de procesos existentes)
            mem_system->ram_frames[frame].pid = process->pid;
            mem_system->ram_frames[frame].page_number = i;
            mem_system->ram_frames[frame].occupied = true;
            mem_system->ram_frames[frame].load_time = time(NULL);

            // Agregar a cola FIFO
            enqueue_fifo(mem_system->fifo_queue, frame);

            process->page_table[i].page_number = i;
            process->page_table[i].frame_number = frame;
            process->page_table[i].state = PAGE_IN_RAM;
            process->page_table[i].valid = true;
            process->page_table[i].modified = false;
            process->page_table[i].swap_position = -1;
            process->page_table[i].last_access = time(NULL);
            process->page_table[i].load_time = time(NULL);

            // Actualizar TLB
            tlb_update(process->pid, i, frame);
```

```
        pages_in_ram++;
    } else {
        // No hay marcos libres en RAM, asignar directamente en Swap
        int swap_frame = find_free_swap_frame();
        if (swap_frame == -1) {
            // No hay espacio ni en RAM ni en Swap
            printf("❌ Error: No hay espacio disponible para el proceso.\n");

            // Liberar páginas ya asignadas
            for (int j = 0; j < i; j++) {
                if (process->page_table[j].state == PAGE_IN_RAM) {
                    int f = process->page_table[j].frame_number;
                    mem_system->ram_frames[f].occupied = false;
                    mem_system->ram_frames[f].pid = -1;
                } else if (process->page_table[j].state == PAGE_IN_SWAP) {
                    int s = process->page_table[j].swap_position;
                    mem_system->swap_frames[s].occupied = false;
                    mem_system->swap_frames[s].pid = -1;
                }
            }

            free(process->page_table);
            free(process);
            return -1;
```

VERDAD, BELLEZA, PROBIDAD

```
            }

            // Asignar directamente en Swap (sin hacer swap out)
            mem_system->swap_frames[swap_frame].pid = process->pid;
            mem_system->swap_frames[swap_frame].page_number = i;
            mem_system->swap_frames[swap_frame].occupied = true;
            mem_system->swap_frames[swap_frame].load_time = time(NULL);

            process->page_table[i].page_number = i;
            process->page_table[i].frame_number = -1;
            process->page_table[i].state = PAGE_IN_SWAP;
            process->page_table[i].valid = false;
            process->page_table[i].modified = false;
            process->page_table[i].swap_position = swap_frame;
            process->page_table[i].last_access = 0;
            process->page_table[i].load_time = time(NULL);
        }
    }

    // Si todas las páginas están en Swap, o algunas están en Swap, marcar proceso
como SWAPPED
    if (pages_in_ram == 0 || (pages_in_ram > 0 && pages_in_ram < num_pages)) {
        process->state = PROC_SWAPPED;
    }

    // Agregar proceso al sistema
```

```
    mem_system->processes[slot] = process;
    mem_system->num_processes++;

    char msg[256];
    snprintf(msg, sizeof(msg),
             "Proceso creado: PID=%d, Nombre='%s', Tamaño=%d KB, Páginas=%d (RAM:%d, Swap:%d)",
             process->pid, process->name, size_kb, num_pages, pages_in_ram, num_pages - pages_in_ram);
    add_log(msg);

    printf("\n✓ Proceso creado exitosamente:\n");
    printf("  PID: %d\n", process->pid);
    printf("  Nombre: %s\n", process->name);
    printf("  Tamaño: %d KB\n", size_kb);
    printf("  Páginas: %d (Tamaño de página: %d KB)\n", num_pages, PAGE_SIZE);
    printf("  Páginas en RAM: %d\n", pages_in_ram);
    printf("  Páginas en Swap: %d\n", num_pages - pages_in_ram);

    if (pages_in_ram < num_pages) {
        printf("  ⚠ Estado: SWAPPED (algunas páginas en swap debido a memoria RAM llena)\n");
    }

    return process->pid;
}
```

VERDAD, BELLEZA, PROBIDAD

```
// Terminar proceso
bool terminate_process(int pid) {
    PCB *process = find_process(pid);
    if (!process) {
        printf("❌ Error: Proceso con PID %d no encontrado.\n", pid);
        return false;
    }

    // Liberar páginas en RAM
    for (int i = 0; i < process->num_pages; i++) {
        if (process->page_table[i].state == PAGE_IN_RAM) {
            int frame = process->page_table[i].frame_number;
            mem_system->ram_frames[frame].occupied = false;
            mem_system->ram_frames[frame].pid = -1;
            mem_system->ram_frames[frame].page_number = -1;
        } else if (process->page_table[i].state == PAGE_IN_SWAP) {
            int swap_frame = process->page_table[i].swap_position;
            mem_system->swap_frames[swap_frame].occupied = false;
            mem_system->swap_frames[swap_frame].pid = -1;
            mem_system->swap_frames[swap_frame].page_number = -1;
        }
    }

    // Invalidar entradas de TLB
```

VERDAD, BELLEZA, PROBIDAD

```
    tlb_invalidate(pid);


    // Actualizar estado
    process->state = PROC_TERMINATED;


    char msg[256];
    snprintf(msg, sizeof(msg), "Proceso terminado: PID=%d, Nombre='%s', Page
Faults=%d",
             process->pid, process->name, process->page_faults);
    add_log(msg);


    // Buscar slot del proceso
    for (int i = 0; i < MAX_PROCESSES; i++) {
        if (mem_system->processes[i] && mem_system->processes[i]->pid == pid) {
            free(mem_system->processes[i]->page_table);
            free(mem_system->processes[i]);
            mem_system->processes[i] = NULL;
            mem_system->num_processes--;
            break;
        }
    }


    printf("✓ Proceso %d terminado y memoria liberada.\n", pid);
    return true;
}
```

VERDAD, BELLEZA, PROBIDAD

```
// Mostrar mapa de memoria

void display_memory_map() {

printf("\n╔═══════════════════════════════════════════════════════════╗\n");

    printf("│                    MAPA DE MEMORIA RAM                    │\n");

printf("╚═══════════════════════════════════════════════════════════╝\n\n");


    int free_frames = 0;

    for (int i = 0; i < NUM_RAM_FRAMES; i++) {

        if (mem_system->ram_frames[i].occupied) {

            printf("  [Marco %2d] Proceso %d, Página %d\n",

                   i,         mem_system->ram_frames[i].pid,         mem_system-
>ram_frames[i].page_number);

        } else {

            printf("  [Marco %2d] [LIBRE]\n", i);

            free_frames++;

        }

    }


    printf("\n  Total marcos RAM: %d | Libres: %d | Ocupados: %d\n",

           NUM_RAM_FRAMES, free_frames, NUM_RAM_FRAMES - free_frames);

    printf("  Utilización: %.2f%%\n",

           ((float)(NUM_RAM_FRAMES - free_frames) / NUM_RAM_FRAMES) * 100);


printf("\n╚═══════════════════════════════════════════════════════════╝\n");
```

VERDAD, BELLEZA, PROBIDAD

```
    printf("│                    MAPA DE ÁREA DE SWAP                    │\n");

printf("└────────────────────────────────────────────────────────────────┘\n\n");

    int free_swap = 0;
    int occupied_swap = 0;

    for (int i = 0; i < NUM_SWAP_FRAMES; i++) {
        if (mem_system->swap_frames[i].occupied) {
            printf("  [Swap %2d] Proceso %d, Página %d\n",
                   i,      mem_system->swap_frames[i].pid,       mem_system-
>swap_frames[i].page_number);
            occupied_swap++;
        } else {
            free_swap++;
        }
    }

    if (occupied_swap == 0) {
        printf("  [Área de Swap vacía]\n");
    }

    printf("\n  Total marcos Swap: %d | Libres: %d | Ocupados: %d\n",
           NUM_SWAP_FRAMES, free_swap, occupied_swap);
    printf("  Utilización: %.2f%%\n",
           ((float)occupied_swap / NUM_SWAP_FRAMES) * 100);
```

VERDAD, BELLEZA, PROBIDAD

```
}


// Mostrar tabla de páginas de un proceso
void display_process_table(int pid) {
    PCB *process = find_process(pid);
    if (!process) {
        printf("❌ Error: Proceso con PID %d no encontrado.\n", pid);
        return;
    }


printf("\n╔═══════════════════════════════════════════════════════════╗\n");
    printf("║             TABLA DE PÁGINAS - Proceso %d                 ║\n",
pid);

printf("╚═══════════════════════════════════════════════════════════╝\n\n");

    printf("  Nombre: %s\n", process->name);
    printf("  Tamaño: %d KB\n", process->size);
    printf("  Número de páginas: %d\n", process->num_pages);
    printf("  Estado: ");

    switch (process->state) {
        case PROC_ACTIVE: printf("ACTIVO\n"); break;
        case PROC_SUSPENDED: printf("SUSPENDIDO\n"); break;
        case PROC_SWAPPED: printf("INTERCAMBIADO (parcial)\n"); break;
```

VERDAD, BELLEZA, PROBIDAD

```
            case PROC_TERMINATED: printf("TERMINADO\n"); break;
        }

        printf("  Page Faults: %d\n\n", process->page_faults);

        printf("  %-8s %-12s %-12s %-8s %-12s\n",
               "Página", "Estado", "Marco RAM", "Válido", "Swap Pos");
        printf("  %s\n", "-------------------------------------------------------------
------");

        for (int i = 0; i < process->num_pages; i++) {
            PageTableEntry *entry = &process->page_table[i];

            printf("  %-8d ", entry->page_number);

            switch (entry->state) {
                case PAGE_IN_RAM:
                    printf("%-12s %-12d %-8s %-12s\n",
                           "EN RAM", entry->frame_number,
                           entry->valid ? "Sí" : "No", "-");
                    break;
                case PAGE_IN_SWAP:
                    printf("%-12s %-12s %-8s %-12d\n",
                           "EN SWAP", "-", "No", entry->swap_position);
                    break;
                default:
```

VERDAD, BELLEZA, PROBIDAD

```
                printf("%-12s %-12s %-8s %-12s\n",
                       "LIBRE", "-", "No", "-");
                break;
            }
        }
    }

// Mostrar estado del sistema
void display_system_status() {

printf("\n╔═══════════════════════════════════════════════════════╗\n");
    printf("║                    ESTADO DEL SISTEMA                 ║\n");
printf("╚═══════════════════════════════════════════════════════╝\n\n");

    printf("  PROCESOS ACTIVOS: %d\n\n", mem_system->num_processes);

    if (mem_system->num_processes == 0) {
        printf("  [No hay procesos activos]\n");
    } else {
        printf("  %-6s %-20s %-12s %-10s %-15s\n",
               "PID", "Nombre", "Tamaño", "Páginas", "Estado");
        printf("  %s\n", "-----------------------------------------------------
----------------");

        for (int i = 0; i < MAX_PROCESSES; i++) {
```

VERDAD, BELLEZA, PROBIDAD

```
            PCB *p = mem_system->processes[i];
            if (p) {
                printf("  %-6d %-20s %-12d %-10d ",
                       p->pid, p->name, p->size, p->num_pages);

                switch (p->state) {
                    case PROC_ACTIVE: printf("%-15s\n", "ACTIVO"); break;
                    case PROC_SUSPENDED: printf("%-15s\n", "SUSPENDIDO"); break;
                    case PROC_SWAPPED: printf("%-15s\n", "INTERCAMBIADO"); break;
                    case PROC_TERMINATED: printf("%-15s\n", "TERMINADO"); break;
                }
            }
        }
    }

    // Resumen de memoria
    int ram_used = 0, swap_used = 0;
    for (int i = 0; i < NUM_RAM_FRAMES; i++) {
        if (mem_system->ram_frames[i].occupied) ram_used++;
    }
    for (int i = 0; i < NUM_SWAP_FRAMES; i++) {
        if (mem_system->swap_frames[i].occupied) swap_used++;
    }

    printf("\n  MEMORIA:\n");
```

VERDAD, BELLEZA, PROBIDAD

```
    printf("  RAM: %d/%d marcos ocupados (%.1f%%)\n",
           ram_used, NUM_RAM_FRAMES,
           ((float)ram_used / NUM_RAM_FRAMES) * 100);
    printf("  Swap: %d/%d marcos ocupados (%.1f%%)\n",
           swap_used, NUM_SWAP_FRAMES,
           ((float)swap_used / NUM_SWAP_FRAMES) * 100);
}

// Mostrar TLB
void display_tlb() {

printf("\n╔═══════════════════════════════════════════════════════════╗\n");
    printf("║           TLB (Translation Lookaside Buffer)            ║\n");

printf("╚═══════════════════════════════════════════════════════════╝\n\n");

    printf("  Tamaño de TLB: %d entradas\n\n", TLB_SIZE);

    printf("  %-10s %-8s %-12s %-12s %-10s\n",
           "Entrada", "PID", "Página", "Marco RAM", "Válido");
    printf("  %s\n", "-----------------------------------------------------------
--");

    for (int i = 0; i < TLB_SIZE; i++) {
        TLBEntry *entry = &mem_system->tlb[i];
```

VERDAD, BELLEZA, PROBIDAD

```
        printf("  %-10d ", i);


        if (entry->valid) {
            printf("%-8d %-12d %-12d %-10s\n",
                   entry->pid, entry->page_number,
                   entry->frame_number, "Sí");
        } else {
            printf("%-8s %-12s %-12s %-10s\n",
                   "-", "-", "-", "No");
        }
    }

    printf("\n  Estadísticas TLB:\n");
    printf("  Aciertos (hits): %d\n", mem_system->total_tlb_hits);
    printf("  Fallos (misses): %d\n", mem_system->total_tlb_misses);

    int total_accesses = mem_system->total_tlb_hits + mem_system-
>total_tlb_misses;
    if (total_accesses > 0) {
        printf("  Tasa de aciertos: %.2f%%\n",
               ((float)mem_system->total_tlb_hits / total_accesses) * 100);
    }
}

// Mostrar estadísticas del sistema
void display_statistics() {
```

VERDAD, BELLEZA, PROBIDAD

```
printf("\n╔═══════════════════════════════════════════════════════════╗\n");
    printf("│                ESTADÍSTICAS DEL SISTEMA                   │\n");

printf("╚═══════════════════════════════════════════════════════════╝\n\n");


    // Calcular fragmentación interna
    int internal_fragmentation = 0;
    for (int i = 0; i < MAX_PROCESSES; i++) {
        PCB *p = mem_system->processes[i];
        if (p) {
            int wasted = (p->num_pages * PAGE_SIZE) - p->size;
            internal_fragmentation += wasted;
        }
    }

    // Calcular utilización de memoria
    int ram_used = 0, swap_used = 0;
    for (int i = 0; i < NUM_RAM_FRAMES; i++) {
        if (mem_system->ram_frames[i].occupied) ram_used++;
    }
    for (int i = 0; i < NUM_SWAP_FRAMES; i++) {
        if (mem_system->swap_frames[i].occupied) swap_used++;
    }

    float ram_utilization = ((float)ram_used / NUM_RAM_FRAMES) * 100;
```

```
    float swap_utilization = ((float)swap_used / NUM_SWAP_FRAMES) * 100;


    // Tiempo promedio de acceso (considerando TLB)
    // Asumiendo: TLB hit = 1ns, TLB miss + RAM = 100ns, Page fault + Swap =
1000ns
    int tlb_accesses = mem_system->total_tlb_hits + mem_system->total_tlb_misses;
    float avg_access_time = 0;

    if (tlb_accesses > 0) {
        float hit_rate = (float)mem_system->total_tlb_hits / tlb_accesses;
        float miss_rate = (float)mem_system->total_tlb_misses / tlb_accesses;
        float    page_fault_rate    =    (float)mem_system->total_page_faults    /
tlb_accesses;

        avg_access_time = (hit_rate * 1) + (miss_rate * 100) + (page_fault_rate *
1000);
    }

    printf("  MÉTRICAS DE RENDIMIENTO:\n\n");

    printf("     %-40s  %d\n",  "Total  de  fallos  de  página:",  mem_system-
>total_page_faults);
    printf("    %-40s  %d\n",  "Total  de  operaciones  de  swap:",  mem_system-
>total_swaps);
    printf("  %-40s %.2f ns\n", "Tiempo promedio de acceso:", avg_access_time);
    printf("  %-40s %d KB\n", "Fragmentación interna:", internal_fragmentation);
```

```
    printf("\n  UTILIZACIÓN DE MEMORIA:\n\n");
    printf("  %-40s %.2f%%\n", "Utilización de RAM:", ram_utilization);
    printf("  %-40s %.2f%%\n", "Utilización de Swap:", swap_utilization);

    printf("\n  TLB:\n\n");
    printf("  %-40s %d\n", "Aciertos en TLB:", mem_system->total_tlb_hits);
    printf("  %-40s %d\n", "Fallos en TLB:", mem_system->total_tlb_misses);

    if (tlb_accesses > 0) {
        printf("  %-40s %.2f%%\n", "Tasa de aciertos en TLB:",
               ((float)mem_system->total_tlb_hits / tlb_accesses) * 100);
    }

    printf("\n  PROCESOS:\n\n");
    printf("  %-40s %d\n", "Procesos activos:", mem_system->num_processes);

    // Tiempo de ejecución
    time_t current_time = time(NULL);
    int runtime = (int)difftime(current_time, mem_system->start_time);
    printf("  %-40s %d segundos\n", "Tiempo de ejecución:", runtime);
}

// Mostrar logs
void display_logs(int count) {

printf("\n╔═══════════════════════════════════════════════════════════════╗\n");
```

VERDAD, BELLEZA, PROBIDAD

```
    printf("│                    REGISTRO DE EVENTOS                    │\n");

printf("└──────────────────────────────────────────────────────────────┘\n\n");

    if (mem_system->log_count == 0) {
        printf("  [No hay eventos registrados]\n");
        return;
    }

    int start = (count > mem_system->log_count) ? 0 : mem_system->log_count -
count;

    printf("  Mostrando últimos %d eventos:\n\n", mem_system->log_count - start);

    for (int i = start; i < mem_system->log_count; i++) {
        LogEntry *entry = &mem_system->logs[i];
        struct tm *timeinfo = localtime(&entry->timestamp);

        printf("  [%02d:%02d:%02d] %s\n",
               timeinfo->tm_hour, timeinfo->tm_min, timeinfo->tm_sec,
               entry->message);
    }
}

// Guardar logs en archivo
void save_logs_to_file(const char *filename) {
```

VERDAD, BELLEZA, PROBIDAD

```
    FILE *file = fopen(filename, "w");
    if (!file) {
        printf("❌ Error: No se pudo crear el archivo de logs.\n");
        return;
    }

    fprintf(file, "========================================\n");
    fprintf(file, "REGISTRO DE EVENTOS DEL SIMULADOR\n");
    fprintf(file, "========================================\n\n");

    for (int i = 0; i < mem_system->log_count; i++) {
        LogEntry *entry = &mem_system->logs[i];
        struct tm *timeinfo = localtime(&entry->timestamp);

        fprintf(file, "[%02d:%02d:%02d] %s\n",
                timeinfo->tm_hour, timeinfo->tm_min, timeinfo->tm_sec,
                entry->message);
    }

    fclose(file);
    printf("✓ Logs guardados en: %s\n", filename);
}

// Obtener entrada de usuario (entero)
int get_user_input_int(const char *prompt) {
```

VERDAD, BELLEZA, PROBIDAD

```
    int value;
    printf("%s", prompt);

    while (scanf("%d", &value) != 1) {
        // Limpiar buffer
        while (getchar() != '\n');
        printf("❌ Entrada inválida. %s", prompt);
    }

    // Limpiar buffer
    while (getchar() != '\n');

    return value;
}

// Limpiar pantalla
void clear_screen() {
    #ifdef _WIN32
        system("cls");
    #else
        system("clear");
    #endif
}

// Pausar pantalla
```

VERDAD, BELLEZA, PROBIDAD

```
void pause_screen() {
    printf("\nPresione ENTER para continuar...");
    getchar();
}

// Mostrar menú principal
void print_menu() {

printf("\n╔═══════════════════════════════════════════════════════════╗\n");
    printf("│        SIMULADOR DE GESTOR DE MEMORIA RAM Y SWAP          │\n");
    printf("╠═══════════════════════════════════════════════════════════╣\n");
    printf("│  GESTIÓN DE PROCESOS:                                     │\n");
    printf("│   1. Crear nuevo proceso                                  │\n");
    printf("│   2. Terminar proceso                                     │\n");
    printf("│   3. Listar procesos activos                              │\n");
    printf("│                                                           │\n");
    printf("│  VISUALIZACIÓN DE MEMORIA:                                │\n");
    printf("│   4. Mostrar mapa de memoria (RAM y Swap)                 │\n");
    printf("│   5. Mostrar tabla de páginas de un proceso               │\n");
    printf("│   6. Mostrar estado de la TLB                             │\n");
    printf("│                                                           │\n");
    printf("│  OPERACIONES DE MEMORIA:                                  │\n");
    printf("│   7. Simular acceso a página (swap in si es necesario)    │\n");
    printf("│                                                           │\n");
    printf("│  INFORMACIÓN Y ESTADÍSTICAS:                              │\n");
    printf("│   8. Ver estadísticas del sistema                         │\n");
```

VERDAD, BELLEZA, PROBIDAD

```
    printf("│   9. Ver registro de eventos (logs)                          │\n");
    printf("│  10. Guardar logs en archivo                                 │\n");
    printf("│                                                              │\n");
    printf("│  11. Salir                                                   │\n");
    printf("└──────────────────────────────────────────────────────────────┘\n");
}

// Ejecutar simulador
void run_simulator() {
    int option;
    bool running = true;

    while (running) {
        print_menu();
        option = get_user_input_int("\nSeleccione una opción: ");

        switch (option) {
            case 1: { // Crear proceso
                clear_screen();

printf("\n══════════════════════════════════════════════════\n");
                printf("              CREAR NUEVO PROCESO\n");

printf("══════════════════════════════════════════════════\n\n");

                char name[32];
```

VERDAD, BELLEZA, PROBIDAD

```
                printf("Ingrese el nombre del proceso: ");
                scanf("%31s", name);
                while (getchar() != '\n'); // Limpiar buffer

                int size = get_user_input_int("Ingrese el tamaño del proceso (KB): ");

                if (size > 0) {
                    create_process(name, size);
                } else {
                    printf("❌ Tamaño inválido.\n");
                }

                pause_screen();
                break;
            }

            case 2: { // Terminar proceso
                clear_screen();
                display_system_status();

                if (mem_system->num_processes == 0) {
                    printf("\n⚠️  No hay procesos activos para terminar.\n");
                } else {
                    int pid = get_user_input_int("\nIngrese el PID del proceso a terminar: ");
```

VERDAD, BELLEZA, PROBIDAD

```
            terminate_process(pid);
        }

        pause_screen();
        break;
    }

    case 3: { // Listar procesos
        clear_screen();
        display_system_status();
        pause_screen();
        break;
    }

    case 4: { // Mapa de memoria
        clear_screen();
        display_memory_map();
        pause_screen();
        break;
    }

    case 5: { // Tabla de páginas
        clear_screen();

        if (mem_system->num_processes == 0) {
```

VERDAD, BELLEZA, PROBIDAD

```
                printf("\n⚠️  No hay procesos activos.\n");
            } else {
                display_system_status();
                int pid = get_user_input_int("\nIngrese el PID del proceso:
");
                display_process_table(pid);
            }

            pause_screen();
            break;
        }

        case 6: { // Mostrar TLB
            clear_screen();
            display_tlb();
            pause_screen();
            break;
        }

        case 7: { // Simular acceso a página
            clear_screen();

            if (mem_system->num_processes == 0) {
                printf("\n⚠️  No hay procesos activos.\n");
                pause_screen();
                break;
```

VERDAD, BELLEZA, PROBIDAD

```
        }

        display_system_status();

        int pid = get_user_input_int("\nIngrese el PID del proceso: ");
        PCB *process = find_process(pid);

        if (!process) {
            printf("❌ Proceso no encontrado.\n");
            pause_screen();
            break;
        }

        int page = get_user_input_int("Ingrese el número de página a acceder: ");

        if (page < 0 || page >= process->num_pages) {
            printf("❌ Número de página inválido (rango: 0-%d).\n", process->num_pages - 1);
            pause_screen();
            break;
        }

        printf("\n--- Simulando acceso a Página %d del Proceso %d ---\n\n", page, pid);
```

VERDAD, BELLEZA, PROBIDAD

```
        // Incrementar accesos a memoria
        mem_system->total_memory_accesses++;

        // Buscar en TLB
        int frame = tlb_lookup(pid, page);

        if (frame != -1) {
            printf("✓ TLB HIT: Página encontrada en TLB (Marco %d)\n",
frame);
            printf("  Acceso directo a memoria física.\n");

            char msg[256];
            snprintf(msg, sizeof(msg), "Acceso a memoria: Proceso %d,
Página %d - TLB HIT (Marco %d)",
                     pid, page, frame);
            add_log(msg);
        } else {
            printf("✗ TLB MISS: Página no encontrada en TLB\n");
            printf("  Consultando tabla de páginas...\n\n");

            PageTableEntry *entry = &process->page_table[page];

            if (entry->state == PAGE_IN_RAM) {
                printf("✓ Página encontrada en RAM (Marco %d)\n", entry-
>frame_number);
                printf("  Actualizando TLB...\n");
```

```
                    tlb_update(pid, page, entry->frame_number);


                    char msg[256];
                    snprintf(msg, sizeof(msg), "Acceso a memoria: Proceso %d, Página %d - En RAM (Marco %d)",
                             pid, page, entry->frame_number);
                    add_log(msg);
                } else if (entry->state == PAGE_IN_SWAP) {
                    printf("✗ PAGE FAULT: Página en Swap (posición %d)\n", entry->swap_position);
                    printf("  Iniciando swap in...\n\n");


                    if (swap_in_page(pid, page)) {
                        printf("✓ Swap in completado exitosamente.\n");
                        printf("  Página ahora en RAM (Marco %d)\n", entry->frame_number);
                        printf("  TLB actualizada.\n");
                    } else {
                        printf("❌ Error al realizar swap in.\n");
                    }
                } else {
                    printf("❌ Página no presente en memoria.\n");
                }
            }
```

VERDAD, BELLEZA, PROBIDAD

```
            pause_screen();
            break;
        }

        case 8: { // Estadísticas
            clear_screen();
            display_statistics();
            pause_screen();
            break;
        }

        case 9: { // Ver logs
            clear_screen();
            int count = get_user_input_int("¿Cuántos eventos desea ver? (0 =
todos): ");

            if (count == 0) {
                count = mem_system->log_count;
            }

            display_logs(count);
            pause_screen();
            break;
        }

        case 10: { // Guardar logs
```

VERDAD, BELLEZA, PROBIDAD

```
        char filename[128];
        time_t now = time(NULL);
        struct tm *t = localtime(&now);

        snprintf(filename, sizeof(filename),
                 "memory_simulator_log_%04d%02d%02d_%02d%02d%02d.txt",
                 t->tm_year + 1900, t->tm_mon + 1, t->tm_mday,
                 t->tm_hour, t->tm_min, t->tm_sec);

        save_logs_to_file(filename);
        pause_screen();
        break;
    }

    case 11: { // Salir
        printf("\n¿Está seguro de que desea salir? (1=Sí, 0=No): ");
        int confirm;
        scanf("%d", &confirm);
        while (getchar() != '\n');

        if (confirm == 1) {
            // Guardar logs automáticamente
            char filename[128];
            time_t now = time(NULL);
            struct tm *t = localtime(&now);
```

VERDAD, BELLEZA, PROBIDAD

```
                snprintf(filename, sizeof(filename),

"memory_simulator_log_%04d%02d%02d_%02d%02d%02d.txt",
                        t->tm_year + 1900, t->tm_mon + 1, t->tm_mday,
                        t->tm_hour, t->tm_min, t->tm_sec);

                save_logs_to_file(filename);

printf("\n╔═══════════════════════════════════════════════════╗\n");
                printf("║                ¡Gracias por usar el simulador!
║\n");
printf("╚═══════════════════════════════════════════════════╝\n\n");

                running = false;
            }
            break;
        }

        default:
            printf("\n❌ Opción inválida. Intente nuevamente.\n");
            pause_screen();
            break;
    }
}
```

VERDAD, BELLEZA, PROBIDAD

```
}

// ==================== FUNCIÓN PRINCIPAL ====================

int main(int argc, char *argv[]) {
    // Cargar configuración
    load_config("config.ini");

    // Inicializar sistema
    init_system();

    printf("\n");
    pause_screen();

    // Ejecutar simulador
    run_simulator();

    // Liberar recursos
    free_system();

    return 0;
}
```

Pruebas

VERDAD, BELLEZA, PROBIDAD

```
Configuracion cargada desde config.ini

+------------------------------------------------------------+
|     INICIALIZANDO SIMULADOR DE GESTOR DE MEMORIA           |
+------------------------------------------------------------+

Memoria RAM inicializada: 2048 KB (8 marcos de 256 KB)
Area de Swap inicializada: 4096 KB (16 marcos de 256 KB)
TLB inicializada: 4 entradas
Algoritmo de reemplazo: FIFO


Presione ENTER para continuar...
```

```
Algoritmo de reemplazo: FIFO


Presione ENTER para continuar...

+------------------------------------------------------------+
|        SIMULADOR DE GESTOR DE MEMORIA RAM Y SWAP           |
+------------------------------------------------------------+
|  GESTION DE PROCESOS:                                      |
|   1. Crear nuevo proceso                                   |
|   2. Terminar proceso                                      |
|   3. Listar procesos activos                               |
|                                                            |
|  VISUALIZACION DE MEMORIA:                                 |
|   4. Mostrar mapa de memoria (RAM y Swap)                  |
|   5. Mostrar tabla de paginas de un proceso                |
|   6. Mostrar estado de la TLB                              |
|                                                            |
|  OPERACIONES DE MEMORIA:                                   |
|   7. Simular acceso a pagina (swap in si es necesario)     |
|                                                            |
|  INFORMACION Y ESTADISTICAS:                               |
|   8. Ver estadisticas del sistema                          |
|   9. Ver registro de eventos (logs)                        |
|  10. Guardar logs en archivo                               |
|                                                            |
|  11. Salir                                                 |
+------------------------------------------------------------+

Seleccione una opcion: 1
```

1. Crear nuevo proceso

```
------------------------------------------------------------
              CREAR NUEVO PROCESO
------------------------------------------------------------

Ingrese el nombre del proceso: P1
Ingrese el tamano del proceso (KB): 1000

 Proceso creado exitosamente:
  PID: 1
  Nombre: P1
  Tamano: 1000 KB
  Paginas: 4 (Tamano de pagina: 256 KB)
  Paginas en RAM: 4
  Paginas en Swap: 0

Presione ENTER para continuar...
```

```
------------------------------------------------------------
              CREAR NUEVO PROCESO
------------------------------------------------------------

Ingrese el nombre del proceso: P2
Ingrese el tamano del proceso (KB): 500

 Proceso creado exitosamente:
  PID: 2
  Nombre: P2
  Tamano: 500 KB
  Paginas: 2 (Tamano de pagina: 256 KB)
  Paginas en RAM: 2
  Paginas en Swap: 0

Presione ENTER para continuar...
```

VERDAD, BELLEZA, PROBIDAD

```
------------------------------------------------------------
              CREAR NUEVO PROCESO
------------------------------------------------------------

Ingrese el nombre del proceso: P3
Ingrese el tamano del proceso (KB): 1200

 Proceso creado exitosamente:
  PID: 3
  Nombre: P3
  Tamano: 1200 KB
  Paginas: 5 (Tamano de pagina: 256 KB)
  Paginas en RAM: 2
  Paginas en Swap: 3
  Estado: SWAPPED (algunas paginas en swap debido a memoria RAM llena)

Presione ENTER para continuar...
```

3. Listar procesos activos

```
+------------------------------------------------------------+
|        SIMULADOR DE GESTOR DE MEMORIA RAM Y SWAP           |
+------------------------------------------------------------+
|  GESTION DE PROCESOS:                                      |
|   1. Crear nuevo proceso                                   |
|   2. Terminar proceso                                      |
|   3. Listar procesos activos                               |
|                                                            |
|  VISUALIZACION DE MEMORIA:                                 |
|   4. Mostrar mapa de memoria (RAM y Swap)                  |
|   5. Mostrar tabla de paginas de un proceso                |
|   6. Mostrar estado de la TLB                              |
|                                                            |
|  OPERACIONES DE MEMORIA:                                   |
|   7. Simular acceso a pagina (swap in si es necesario)     |
|                                                            |
|  INFORMACION Y ESTADISTICAS:                               |
|   8. Ver estadisticas del sistema                          |
|   9. Ver registro de eventos (logs)                        |
|  10. Guardar logs en archivo                               |
|                                                            |
|  11. Salir                                                 |
+------------------------------------------------------------+

Seleccione una opcion: 3
```

```
+------------------------------------------------------------+
|                    ESTADO DEL SISTEMA                      |
+------------------------------------------------------------+

  PROCESOS ACTIVOS: 3

  PID    Nombre               Tamano       Paginas    Estado
  ---------------------------------------------------------------------
  1      P1                   1000         4          ACTIVO
  2      P2                   500          2          ACTIVO
  3      P3                   1200         5          INTERCAMBIADO

  MEMORIA:
  RAM: 8/8 marcos ocupados (100.0%)
  Swap: 3/16 marcos ocupados (18.8%)

Presione ENTER para continuar...
```

4. Mostrar mapa de memoria (RAM y Swap)

```
+-------------------------------------------------------------+
|                      MAPA DE MEMORIA RAM                    |
+-------------------------------------------------------------+

  [Marco  0] Proceso 1, Pagina 0
  [Marco  1] Proceso 1, Pagina 1
  [Marco  2] Proceso 1, Pagina 2
  [Marco  3] Proceso 1, Pagina 3
  [Marco  4] Proceso 2, Pagina 0
  [Marco  5] Proceso 2, Pagina 1
  [Marco  6] Proceso 3, Pagina 0
  [Marco  7] Proceso 3, Pagina 1

  Total marcos RAM: 8 | Libres: 0 | Ocupados: 8
  Utilizacion: 100.00%

+-------------------------------------------------------------+
|                     MAPA DE AREA DE SWAP                    |
+-------------------------------------------------------------+

  [Swap  0] Proceso 3, Pagina 2
  [Swap  1] Proceso 3, Pagina 3
  [Swap  2] Proceso 3, Pagina 4

  Total marcos Swap: 16 | Libres: 13 | Ocupados: 3
  Utilizacion: 18.75%

Presione ENTER para continuar...
```

5. Mostrar tabla de páginas de un proceso

```
+------------------------------------------------------------+
|                    ESTADO DEL SISTEMA                      |
+------------------------------------------------------------+

  PROCESOS ACTIVOS: 3

  PID    Nombre               Tamano       Paginas    Estado
  ---------------------------------------------------------------------
  1      P1                   1000         4          ACTIVO
  2      P2                   500          2          ACTIVO
  3      P3                   1200         5          INTERCAMBIADO

  MEMORIA:
  RAM: 8/8 marcos ocupados (100.0%)
  Swap: 3/16 marcos ocupados (18.8%)

Ingrese el PID del proceso: 1
```

```
+------------------------------------------------------------+
|              TABLA DE PAGINAS - Proceso 1                  |
+------------------------------------------------------------+

  Nombre: P1
  Tamano: 1000 KB
  Numero de paginas: 4
  Estado: ACTIVO
  Page Faults: 0

  Pagina   Estado       Marco RAM    Valido   Swap Pos
  ---------------------------------------------------------------
  0        EN RAM       0            Si       -
  1        EN RAM       1            Si       -
  2        EN RAM       2            Si       -
  3        EN RAM       3            Si       -

Presione ENTER para continuar...
```

VERDAD, BELLEZA, PROBIDAD

6. Mostrar estado de la TLB

```
+------------------------------------------------------------+
|           TLB (Translation Lookaside Buffer)               |
+------------------------------------------------------------+

  Tamano de TLB: 4 entradas

  Entrada    PID      Pagina       Marco RAM    Valido
  ------------------------------------------------------------
  0          2        0            4            Si
  1          2        1            5            Si
  2          3        0            6            Si
  3          3        1            7            Si

  Estadisticas TLB:
  Aciertos (hits): 0
  Fallos (misses): 0

Presione ENTER para continuar...
```

7. Simular acceso a página (swap in si es necesario)

```
+------------------------------------------------------------+
|                   ESTADO DEL SISTEMA                       |
+------------------------------------------------------------+

  PROCESOS ACTIVOS: 3

  PID    Nombre                Tamano       Paginas    Estado
  ------------------------------------------------------------------------
  1      P1                    1000         4          ACTIVO
  2      P2                    500          2          ACTIVO
  3      P3                    1200         5          INTERCAMBIADO

  MEMORIA:
  RAM: 8/8 marcos ocupados (100.0%)
  Swap: 3/16 marcos ocupados (18.8%)

Ingrese el PID del proceso: 2
Ingrese el numero de pagina a acceder: 2
Numero de pagina invalido (rango: 0-1).

Presione ENTER para continuar...
```

```
+------------------------------------------------------------+
|                    ESTADO DEL SISTEMA                      |
+------------------------------------------------------------+

  PROCESOS ACTIVOS: 3

  PID    Nombre               Tamano       Paginas    Estado
  ---------------------------------------------------------------------------
  1      P1                   1000         4          ACTIVO
  2      P2                   500          2          ACTIVO
  3      P3                   1200         5          INTERCAMBIADO

  MEMORIA:
  RAM: 8/8 marcos ocupados (100.0%)
  Swap: 3/16 marcos ocupados (18.8%)

Ingrese el PID del proceso: 2
Ingrese el numero de pagina a acceder: 1

--- Simulando acceso a Pagina 1 del Proceso 2 ---

 TLB HIT: Pagina encontrada en TLB (Marco 5)
  Acceso directo a memoria fisica.

Presione ENTER para continuar...
```

2. Terminar proceso

```
+------------------------------------------------------------+
|                    ESTADO DEL SISTEMA                      |
+------------------------------------------------------------+

  PROCESOS ACTIVOS: 3

  PID    Nombre               Tamano       Paginas    Estado
  ---------------------------------------------------------------------------
  1      P1                   1000         4          ACTIVO
  2      P2                   500          2          ACTIVO
  3      P3                   1200         5          INTERCAMBIADO

  MEMORIA:
  RAM: 8/8 marcos ocupados (100.0%)
  Swap: 3/16 marcos ocupados (18.8%)

Ingrese el PID del proceso a terminar: 2
 Proceso 2 terminado y memoria liberada.

Presione ENTER para continuar...
```

8. Ver estadísticas del sistema

```
+------------------------------------------------------------+
|                ESTADISTICAS DEL SISTEMA                    |
+------------------------------------------------------------+

  M├ëTRICAS DE RENDIMIENTO:

  Total de fallos de pagina:                0
  Total de operaciones de swap:             0
  Tiempo promedio de acceso:                1.00 ns
  Fragmentacion interna:                    104 KB

  UTILIZACI├ôN DE MEMORIA:

  Utilizacion de RAM:                       75.00%
  Utilizacion de Swap:                      18.75%

  TLB:

  Aciertos en TLB:                          1
  Fallos en TLB:                            0
  Tasa de aciertos en TLB:                  100.00%

  PROCESOS:

  Procesos activos:                         2
  Tiempo de ejecucion:                      667 segundos

Presione ENTER para continuar...
```

9. Ver registro de eventos (logs)

```
Cuantos eventos desea ver? (0 = todos): 0

+------------------------------------------------------------+
|                   REGISTRO DE EVENTOS                      |
+------------------------------------------------------------+

  Mostrando ultimos 6 eventos:

  [19:02:05] Sistema de memoria inicializado correctamente
  [19:02:31] Proceso creado: PID=1, Nombre='P1', Tamano=1000 KB, Paginas=4 (RAM:4, Swap:0)
  [19:03:01] Proceso creado: PID=2, Nombre='P2', Tamano=500 KB, Paginas=2 (RAM:2, Swap:0)
  [19:03:29] Proceso creado: PID=3, Nombre='P3', Tamano=1200 KB, Paginas=5 (RAM:2, Swap:3)
  [19:11:32] Acceso a memoria: Proceso 2, Pagina 1 - TLB HIT (Marco 5)
  [19:12:37] Proceso terminado: PID=2, Nombre='P2', Page Faults=0

Presione ENTER para continuar...
```

10. Guardar logs en archivo

```
+------------------------------------------------------------+
|        SIMULADOR DE GESTOR DE MEMORIA RAM Y SWAP           |
+------------------------------------------------------------+
|  GESTION DE PROCESOS:                                      |
|   1. Crear nuevo proceso                                   |
|   2. Terminar proceso                                      |
|   3. Listar procesos activos                               |
|                                                            |
|  VISUALIZACION DE MEMORIA:                                 |
|   4. Mostrar mapa de memoria (RAM y Swap)                  |
|   5. Mostrar tabla de paginas de un proceso                |
|   6. Mostrar estado de la TLB                              |
|                                                            |
|  OPERACIONES DE MEMORIA:                                   |
|   7. Simular acceso a pagina (swap in si es necesario)     |
|                                                            |
|  INFORMACION Y ESTADISTICAS:                               |
|   8. Ver estadisticas del sistema                          |
|   9. Ver registro de eventos (logs)                        |
|  10. Guardar logs en archivo                               |
|                                                            |
|  11. Salir                                                 |
+------------------------------------------------------------+

Seleccione una opcion: 10
 Logs guardados en: memory_simulator_log_20251202_191423.txt

Presione ENTER para continuar...
```

11. Salir

```
+------------------------------------------------------------+
|        SIMULADOR DE GESTOR DE MEMORIA RAM Y SWAP           |
+------------------------------------------------------------+
|  GESTION DE PROCESOS:                                      |
|   1. Crear nuevo proceso                                   |
|   2. Terminar proceso                                      |
|   3. Listar procesos activos                               |
|                                                            |
|  VISUALIZACION DE MEMORIA:                                 |
|   4. Mostrar mapa de memoria (RAM y Swap)                  |
|   5. Mostrar tabla de paginas de un proceso                |
|   6. Mostrar estado de la TLB                              |
|                                                            |
|  OPERACIONES DE MEMORIA:                                   |
|   7. Simular acceso a pagina (swap in si es necesario)     |
|                                                            |
|  INFORMACION Y ESTADISTICAS:                               |
|   8. Ver estadisticas del sistema                          |
|   9. Ver registro de eventos (logs)                        |
|  10. Guardar logs en archivo                               |
|                                                            |
|  11. Salir                                                 |
+------------------------------------------------------------+

Seleccione una opcion: 11

Esta seguro de que desea salir? (1=Si, 0=No):
```