SIMULADOR DE GESTOR DE MEMORIA

RAM y Área de Intercambio (Swap)

MANUAL TÉCNICO

Materia: Sistemas Operativos
Institución: Universidad Autónoma de Tamaulipas
Profesor: Dante Adolfo Muñoz Quintero
Semestre: 7°

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

Fecha: Diciembre 2025

Tabla de Contenidos

1. Introducción

1.1 Propósito del Documento

Este manual técnico describe la arquitectura, implementación y funcionamiento interno del Simulador de Gestor de Memoria RAM y Swap desarrollado para la materia de Sistemas Operativos. El documento está dirigido a desarrolladores, estudiantes y profesores que deseen comprender los detalles técnicos del simulador.

1.2 Alcance

El simulador implementa un sistema completo de gestión de memoria basado en paginación, incluyendo:

- Memoria física (RAM) dividida en marcos de tamaño fijo
- Área de intercambio (Swap) para memoria virtual
- Translation Lookaside Buffer (TLB) para acelerar traducciones
- Algoritmo de reemplazo FIFO (First-In, First-Out)
- Operaciones de swapping (swap in/out)
- Sistema de logs y métricas de rendimiento

1.3 Requisitos Técnicos

El simulador fue desarrollado con las siguientes especificaciones:

- Lenguaje: C (estándar C99)
- Compilador: GCC con flags -Wall -Wextra -O2
- Sistema operativo: Compatible con Linux/Unix y Windows (MinGW)
- Librerías estándar: stdio.h, stdlib.h, string.h, time.h, stdbool.h

2. Arquitectura del Sistema

2.1 Componentes Principales

El simulador está compuesto por los siguientes módulos principales:

Sistema de Memoria (MemorySystem): Estructura central que contiene todos los componentes: marcos de RAM y Swap, procesos activos, TLB, cola FIFO, logs y estadísticas.

Gestor de Procesos (PCB): Maneja la creación, terminación y estado de los procesos. Cada proceso tiene su propia tabla de páginas.

Tabla de Páginas (PageTableEntry[]): Mapea páginas lógicas a marcos físicos. Contiene información sobre estado, validez, modificación y ubicación de cada página.

TLB (Translation Lookaside Buffer): Cache de traducciones página→marco para acelerar accesos a memoria. Usa LRU para reemplazo.

Cola FIFO (FIFOQueue): Implementa el algoritmo de reemplazo FIFO. Mantiene el orden de llegada de páginas a RAM.

Sistema de Logs (LogEntry[]): Registra todos los eventos del sistema con timestamps para análisis y debugging.

2.2 Estructuras de Datos

El simulador utiliza las siguientes estructuras de datos principales:

Frame (Marco de memoria)

```
typedef struct {
    int pid;            // PID del proceso (-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;
```

PageTableEntry (Entrada de tabla de páginas)

```
typedef struct {
    int page_number;    // Número de página lógica
    int frame_number;   // Marco físico (-1 si no en RAM)
    PageState state;    // Estado (RAM/Swap/Free)
    bool valid;         // Bit de validez
```

```
    bool modified;       // Bit de modificación
    int swap_position;   // Posición en Swap
    time_t last_access;  // Último acceso (LRU)
    time_t load_time;    // Carga (FIFO)
} PageTableEntry;
```

PCB (Process Control Block)

```
typedef struct {
    int pid;              // ID del proceso
    char name[32];        // Nombre
    int size;             // Tamaño en KB
    int num_pages;        // Páginas necesarias
    ProcessState state;   // Estado del proceso
    PageTableEntry *page_table; // Tabla de páginas
    time_t creation_time;    // Creación
    int page_faults;      // Fallos de página
} PCB;
```

TLBEntry (Entrada de TLB)

```
typedef struct {
    int pid;          // PID del proceso
    int page_number;  // Número de página
    int frame_number; // Marco físico
    bool valid;       // Entrada válida
    time_t last_access;  // Para reemplazo LRU
} TLBEntry;
```

2.3 Flujo de Datos

El flujo principal de datos en el simulador sigue este patrón:

13. 1. Usuario crea proceso → Sistema calcula páginas necesarias
14. 2. Sistema asigna páginas en RAM (si hay espacio disponible)
15. 3. Si RAM llena → Selecciona víctima con FIFO → Swap out
16. 4. Páginas restantes se asignan en Swap
17. 5. Usuario simula acceso a página → Busca en TLB
18. 6. Si TLB miss → Consulta tabla de páginas
19. 7. Si página en Swap → Page fault → Swap in
20. 8. Actualiza TLB con traducción exitosa

3. Descripción de Módulos

3.1 Sistema de Inicialización

La función init_system() inicializa todos los componentes del simulador:

- Calcula número de marcos en RAM y Swap basándose en tamaños configurados
- Asigna memoria para arrays de marcos (ram_frames y swap_frames)
- Inicializa cada marco como libre (pid = -1, occupied = false)
- Crea array de procesos (inicialmente NULL)
- Inicializa TLB con entradas inválidas
- Crea cola FIFO con capacidad = número de marcos RAM
- Asigna buffer de logs (MAX_LOG_ENTRIES)
- Inicializa estadísticas a cero
- Registra timestamp de inicio

Función principal:

```
void init_system() {
    NUM_RAM_FRAMES = RAM_SIZE / PAGE_SIZE;
    NUM_SWAP_FRAMES = SWAP_SIZE / PAGE_SIZE;
    mem_system =
(MemorySystem*)malloc(sizeof(MemorySystem));
    // ... inicialización de componentes ...
}
```

3.2 Gestión de Memoria

El módulo de gestión de memoria es responsable de:

- Asignación de páginas en RAM y Swap
- Búsqueda de marcos libres
- Actualización de tablas de páginas
- Liberación de memoria al terminar procesos

Funciones principales:

int allocate_page_in_ram(int pid, int page_number)
Asigna una página en RAM. Si no hay espacio, invoca swap_out.

int find_free_ram_frame()
Busca el primer marco libre en RAM. Retorna -1 si no hay.

int find_free_swap_frame()
Busca el primer marco libre en Swap. Retorna -1 si no hay.

3.3 Translation Lookaside Buffer (TLB)

El TLB es una cache asociativa que almacena las traducciones página→marco más recientes. Reduce significativamente el tiempo de acceso a memoria.

Características:

- Tamaño configurable (por defecto 4 entradas)
- Algoritmo de reemplazo LRU (Least Recently Used)
- Invalidación automática al terminar procesos
- Actualización tras cada traducción exitosa

Operaciones:

int tlb_lookup(int pid, int page_number)
Busca traducción en TLB. Retorna marco o -1 si miss. Actualiza last_access.

void tlb_update(int pid, int page_number, int frame_number)
Agrega/actualiza entrada en TLB. Reemplaza entrada más antigua si está lleno.

void tlb_invalidate(int pid)
Invalida todas las entradas de un proceso (usado al terminar).

3.4 Algoritmo FIFO

El algoritmo FIFO (First-In, First-Out) se implementa mediante una cola circular que mantiene el orden de llegada de páginas a la RAM.

Estructura de la cola:

```
typedef struct {
    int *queue;      // Array de índices de marcos
    int front;       // Frente de la cola
    int rear;        // Final de la cola
    int size;        // Tamaño actual
    int capacity;    // Capacidad máxima
} FIFOQueue;
```

Operaciones:

- enqueue_fifo(queue, frame_index): Agrega marco al final
- dequeue_fifo(queue): Remueve y retorna marco más antiguo
- is_fifo_empty(queue): Verifica si la cola está vacía

Cuando se necesita un marco y la RAM está llena, se llama a select_victim_page_fifo() que retorna el índice del marco al frente de la cola (el más antiguo).

3.5 Swapping

El sistema de swapping implementa memoria virtual permitiendo que procesos más grandes que la RAM física puedan ejecutarse. Incluye dos operaciones fundamentales:

Swap Out (Desalojar página):

Mueve una página de RAM al área de Swap cuando se necesita liberar espacio.

- Verifica que el marco esté ocupado
- Busca proceso y entrada de tabla de páginas
- Busca marco libre en Swap
- Copia información del marco a Swap
- Actualiza tabla de páginas (state=PAGE_IN_SWAP, valid=false)
- Libera marco en RAM
- Invalida entrada en TLB
- Incrementa contador de swaps
- Registra evento en logs

Swap In (Traer página):

Trae una página del área de Swap a RAM cuando se produce un page fault.

- Verifica que página esté en Swap
- Busca marco libre en RAM
- Si RAM llena: selecciona víctima con FIFO y hace swap_out
- Copia página de Swap a RAM
- Actualiza tabla de páginas (state=PAGE_IN_RAM, valid=true)
- Libera marco en Swap
- Actualiza TLB con nueva traducción
- Agrega marco a cola FIFO
- Incrementa contador de page faults
- Registra evento en logs

4. Implementación de Algoritmos

4.1 Algoritmo de Paginación

Pseudocódigo del algoritmo de paginación al crear proceso:

```
ALGORITMO: crear_proceso(nombre, tamaño_kb)
1. num_páginas ← ⌈tamaño_kb / TAMAÑO_PÁGINA⌉
2. Verificar espacio disponible en RAM + Swap
3. Crear PCB con información del proceso
4. Crear tabla_páginas[num_páginas]
5. PARA cada página i de 0 hasta num_páginas-1:
   a. marco ← asignar_página_en_ram(pid, i)
   b. SI marco != -1 ENTONCES:
      - tabla_páginas[i].estado ← PAGE_IN_RAM
      - tabla_páginas[i].marco ← marco
      - tabla_páginas[i].válido ← true
      - Actualizar TLB
     SINO:
      - marco_swap ← buscar_marco_libre_swap()
      - tabla_páginas[i].estado ← PAGE_IN_SWAP
      - tabla_páginas[i].swap_pos ← marco_swap
      - tabla_páginas[i].válido ← false
6. Agregar proceso al sistema
7. Registrar en logs
```

4.2 Algoritmo FIFO para Reemplazo

Pseudocódigo del algoritmo FIFO:

```
ALGORITMO: seleccionar_víctima_fifo()
1. SI cola_fifo está vacía ENTONCES:
   RETORNAR -1
2. víctima ← desencolar(cola_fifo)
3. RETORNAR víctima

ALGORITMO: asignar_página_en_ram(pid, num_página)
1. marco ← buscar_marco_libre_ram()
2. SI marco == -1 ENTONCES:
   a. víctima ← seleccionar_víctima_fifo()
   b. SI víctima == -1 ENTONCES RETORNAR -1
   c. swap_out(víctima)
   d. marco ← víctima
3. ram_frames[marco].pid ← pid
4. ram_frames[marco].página ← num_página
5. ram_frames[marco].ocupado ← true
```

6. ram_frames[marco].tiempo_carga ← tiempo_actual()
7. encolar(cola_fifo, marco)
8. RETORNAR marco

4.3 Algoritmo de Swap Out

Pseudocódigo de swap out:

ALGORITMO: swap_out(índice_marco)
1. SI marco no válido ENTONCES RETORNAR false
2. proceso ← buscar_proceso(marco.pid)
3. entrada_página ← proceso.tabla_páginas[marco.num_página]
4. marco_swap ← buscar_marco_libre_swap()
5. SI marco_swap == -1 ENTONCES:
- Registrar error (Swap lleno)
- RETORNAR false
6. swap_frames[marco_swap] ← ram_frames[índice_marco]
7. entrada_página.estado ← PAGE_IN_SWAP
8. entrada_página.marco ← -1
9. entrada_página.swap_pos ← marco_swap
10. entrada_página.válido ← false
11. ram_frames[índice_marco].ocupado ← false
12. tlb_invalidar(proceso.pid)
13. total_swaps++
14. Registrar en logs
15. RETORNAR true

4.4 Algoritmo de Swap In

Pseudocódigo de swap in:

ALGORITMO: swap_in(pid, num_página)
1. proceso ← buscar_proceso(pid)
2. entrada_página ← proceso.tabla_páginas[num_página]
3. SI entrada_página.estado != PAGE_IN_SWAP ENTONCES:
RETORNAR false
4. marco_ram ← buscar_marco_libre_ram()
5. SI marco_ram == -1 ENTONCES:
a. víctima ← seleccionar_víctima_fifo()
b. SI víctima == -1 ENTONCES RETORNAR false
c. swap_out(víctima)
d. marco_ram ← víctima
6. ram_frames[marco_ram] ←
swap_frames[entrada_página.swap_pos]
7. entrada_página.estado ← PAGE_IN_RAM

8. entrada_página.marco ← marco_ram
9. entrada_página.swap_pos ← -1
10. entrada_página.válido ← true
11. swap_frames[posición_swap].ocupado ← false
12. tlb_actualizar(pid, num_página, marco_ram)
13. encolar(cola_fifo, marco_ram)
14. proceso.page_faults++
15. total_page_faults++
16. total_swaps++
17. Registrar en logs
18. RETORNAR true

---

5. API y Funciones Principales

void load_config(const char *filename)
Carga configuración desde archivo config.ini. Lee parámetros de RAM, Swap, página y TLB.

void init_system()
Inicializa todos los componentes del sistema de memoria.

void free_system()
Libera toda la memoria asignada al sistema.

int create_process(const char *name, int size_kb)
Crea un nuevo proceso y asigna sus páginas. Retorna PID o -1 si falla.

bool terminate_process(int pid)
Termina un proceso y libera sus recursos. Retorna true si exitoso.

PCB* find_process(int pid)
Busca y retorna PCB de un proceso. NULL si no existe.

int allocate_page_in_ram(int pid, int page_number)
Asigna una página en RAM. Retorna índice de marco o -1.

bool swap_out_page(int frame_index)
Mueve página de RAM a Swap. Retorna true si exitoso.

bool swap_in_page(int pid, int page_number)
Trae página de Swap a RAM. Retorna true si exitoso.

int tlb_lookup(int pid, int page_number)
Busca traducción en TLB. Retorna marco o -1.

void tlb_update(int pid, int page_number, int frame_number)
Actualiza o agrega entrada en TLB.

void display_memory_map()
Muestra estado completo de RAM y Swap.

void display_process_table(int pid)
Muestra tabla de páginas de un proceso.

void display_statistics()
Muestra métricas de rendimiento del sistema.

void add_log(const char *message)
Agrega entrada al sistema de logs con timestamp.

void save_logs_to_file(const char *filename)
Exporta logs a archivo de texto.

6. Configuración del Sistema

El simulador se configura mediante el archivo config.ini:

```
[MEMORIA]
RAM_SIZE = 2048          # Tamaño de RAM en KB
SWAP_SIZE = 4096         # Tamaño de Swap en KB
PAGE_SIZE = 256          # Tamaño de página en KB

[TLB]
TLB_SIZE = 4             # Entradas en TLB

[SISTEMA]
MAX_PROCESSES = 50       # Máximo de procesos
REPLACEMENT_ALGORITHM = FIFO
VERBOSE_LOGS = 1         # Logs detallados
```

Parámetros configurables:

RAM_SIZE: Tamaño total de memoria física en KB. Debe ser múltiplo de PAGE_SIZE.

SWAP_SIZE: Tamaño total del área de intercambio en KB. Debe ser múltiplo de PAGE_SIZE.

PAGE_SIZE: Tamaño de página/marco en KB. Determina granularidad de asignación.

TLB_SIZE: Número de entradas en el TLB. Mayor tamaño mejora hit rate pero usa más memoria.

MAX_PROCESSES: Máximo número de procesos simultáneos permitidos.

Nota: Si el archivo config.ini no existe, el sistema usa valores por defecto y muestra una advertencia al iniciar.

7. Sistema de Logs

El sistema mantiene un registro detallado de todos los eventos importantes:

- Timestamp preciso para cada evento
- Máximo de 1000 entradas almacenadas
- Exportación automática a archivos de texto
- Formato legible para análisis y debugging

Eventos registrados:

- Inicialización del sistema
- Creación y terminación de procesos
- Asignación de páginas en RAM y Swap
- Operaciones de swap out/in
- Accesos a memoria (TLB hit/miss)
- Page faults
- Errores (memoria llena, proceso no encontrado, etc.)

Formato de log:

> ***[HH:MM:SS] Mensaje del evento***
>
> ***Ejemplos:***
> ***[14:30:25] Sistema de memoria inicializado correctamente***
> ***[14:30:32] Proceso creado: PID=1, Nombre='Editor', Tamaño=600 KB***
> ***[14:31:02] SWAP OUT: Proceso 1, Página 2 movida de RAM[7] a Swap[0]***
> ***[14:31:15] Acceso a memoria: Proceso 2, Página 1 - TLB HIT (Marco 5)***

---

8. Métricas y Estadísticas

El simulador calcula y mantiene las siguientes métricas de rendimiento:

Total de fallos de página (Page Faults): Número de veces que se accedió a una página en Swap. Indica eficiencia del sistema.

Total de operaciones de swap: Suma de swap_in y swap_out. Alto valor indica presión de memoria.

Tiempo promedio de acceso: Calculado como: (hit_rate × 1ns) + (miss_rate × 100ns) + (pf_rate × 1000ns)

Fragmentación interna: Espacio desperdiciado en páginas parcialmente ocupadas. En KB.

Utilización de RAM: Porcentaje de marcos ocupados en RAM. Indica carga del sistema.

Utilización de Swap: Porcentaje de marcos ocupados en Swap. Swap alto indica problema de memoria.

Aciertos en TLB (TLB Hits): Número de traducciones encontradas en TLB. Mayor es mejor.

Fallos en TLB (TLB Misses): Número de consultas a tabla de páginas. Menor es mejor.

Tasa de aciertos en TLB: Porcentaje: (hits / (hits + misses)) × 100. Objetivo: > 80%.

Cálculo del tiempo promedio de acceso:

tlb_accesses = total_tlb_hits + total_tlb_misses
hit_rate = total_tlb_hits / tlb_accesses
miss_rate = total_tlb_misses / tlb_accesses
pf_rate = total_page_faults / tlb_accesses

avg_access_time = (hit_rate × 1) + (miss_rate × 100) + (pf_rate × 1000)

Donde:
- TLB hit = 1 nanosegundo
- TLB miss + RAM = 100 nanosegundos
- Page fault + Swap = 1000 nanosegundos

9. Compilación y Despliegue

9.1 Compilación Manual

gcc -Wall -Wextra -std=c99 -O2 -o memory_simulator memory_simulator.c

---

9.2 Compilación con Make

El Makefile incluido proporciona varios comandos:

make: Compilar el simulador

make run: Compilar y ejecutar

make clean: Eliminar archivos generados

make dirs: Crear estructura de directorios

make help: Mostrar ayuda

9.3 Flags de Compilación

-Wall: Habilitar todos los warnings comunes

-Wextra: Habilitar warnings adicionales

-std=c99: Usar estándar C99

-O2: Optimización nivel 2 (balance velocidad/tamaño)

9.4 Dependencias

El simulador solo requiere la biblioteca estándar de C:

- stdio.h
- stdlib.h
- string.h
- time.h
- stdbool.h

No se requieren librerías externas ni configuraciones especiales.

10. Limitaciones y Consideraciones

10.1 Limitaciones del Sistema

Máximo de procesos: Limitado a 50 procesos simultáneos (MAX_PROCESSES)

Buffer de logs: Máximo 1000 entradas (MAX_LOG_ENTRIES). Logs más antiguos se pierden.

Simulación paso a paso: No es tiempo real. Usuario controla cada acción.

Sin multiprocesamiento: No simula ejecución concurrente de procesos.

TLB simplificado: Implementación básica. TLB real es más complejo.

Sin protección de memoria: No simula permisos ni violaciones de acceso.

Sin segmentación: Solo implementa paginación, no esquemas híbridos.

Proceso único activo: No simula context switching entre procesos.

10.2 Consideraciones de Diseño

Decisiones importantes de diseño:

- Uso de FIFO por simplicidad. LRU requeriría más overhead.
- Páginas de tamaño fijo para simplificar asignación.
- TLB con LRU para reflejar hardware real.
- Logs con timestamp para facilitar debugging.
- Configuración externa (config.ini) para flexibilidad.
- Sin persistencia de estado (se pierde al salir).

10.3 Mejoras Futuras

- Implementar algoritmos adicionales: LRU, Reloj, Óptimo
- Agregar GUI gráfica con visualización en tiempo real
- Simular ejecución concurrente de procesos
- Implementar protección de memoria y permisos
- Soporte para paginación multinivel
- Memoria compartida entre procesos
- Simulación de working set
- Exportación de estadísticas a CSV/JSON

11. Conclusiones

Este simulador proporciona una implementación completa y funcional de un gestor de memoria basado en paginación con soporte para memoria virtual. Los principales logros incluyen:

- Implementación correcta del algoritmo de paginación
- Sistema de swapping funcional con FIFO
- TLB operativo que mejora el rendimiento
- Visualización clara del estado de la memoria
- Métricas precisas de rendimiento
- Sistema robusto de logs para análisis
- Código modular y bien documentado
- Interfaz de usuario intuitiva

El proyecto cumple con todos los requerimientos establecidos y proporciona una herramienta educativa valiosa para comprender el funcionamiento interno de los gestores de memoria en sistemas operativos modernos.

La implementación en C permite apreciar los detalles de bajo nivel de la gestión de memoria, mientras que la estructura modular facilita futuras extensiones y mejoras.

Apéndices

Apéndice A: Glosario de Términos

Frame (Marco): Bloque de memoria física de tamaño fijo

Page (Página): Bloque de memoria lógica de tamaño fijo

Page Table (Tabla de páginas): Estructura que mapea páginas a marcos

TLB: Translation Lookaside Buffer - Cache de traducciones

Page Fault: Fallo de página - Acceso a página no presente en RAM

Swap: Área de disco usada para memoria virtual

Swap In: Traer página de disco a RAM

Swap Out: Mover página de RAM a disco

FIFO: First-In First-Out - Algoritmo de reemplazo

PCB: Process Control Block - Bloque de control de proceso

Valid Bit: Bit de validez - Indica si página está en RAM

Dirty Bit: Bit de modificación - Indica si página fue modificada

Apéndice B: Estructura de Archivos

```
Producto_Integrador_SO/
├── simulador_memoria.c     # Código fuente principal
├── config.ini              # Configuración
├── Makefile                # Script de compilación
├── README.md               # Documentación
├── src/                    # Código fuente
│   ├── simulador_memoria.c
│   └── config.ini
├── docs/                   # Documentación
│   ├── manual_tecnico.pdf
│   ├── manual_usuario.pdf
└── tests/                  # Pruebas
    ├── documento_pruebas.pdf
    └── *.txt               # Archivos de logs
```

Apéndice C: Referencias

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.
- Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems (4th ed.). Pearson.
- Stallings, W. (2018). Operating Systems: Internals and Design Principles (9th ed.). Pearson.