Sistemas Operativos Segundo Semestre 2025

Prof. Cecilia Hernández

Tarea 2: Barrera reutilizable y simulación de memoria

Fecha de entrega: 01/12/2025

Integrantes:

- Matías Concha Aguilera
- Benjamín Díaz
- Roberto Cruz Pinto

Parte 1, barrera reutilizable:

Flujo del trabajo e implementación:

- Se comenzó por la definición de las variables (cantidad de hebras y etapas) junto con la función de trabajo de cada hebra (Por simpleza se usa un sleep de un tiempo random). Cada hebra se guarda en un vector por comodidad de acceso.
- Por enunciado, se simula un monitor usando mutex y variables de condición, estas 2 primitivas se usan para:
    - Mutex: Una barrera debe lograr que varias hebras se sincronicen en la misma "etapa", pero una condición de carrera evidente de esta situación surge cuando el contador de hebras activas/esperando no se accede o actualiza correctamente, provocando Deadlocks o un salto de etapas. El mutex está para atomizar tanto los cambios en el contador de hebras como en el estado actual de estas (Para que wait y broadcast funcionen correctamente)

    - Variables de condición: Se usará para controlar la actividad de las hebras, con tal de detener cualquier hebra que no sea la última, en el caso de que si sea la última, se usa un broadcast que logra despertar a todas y cada una de las hebras que estaban esperando.

- Para crear el monitor, se usan las primitivas definidas junto con variables de contador de hebras activas y etapa actual, usando while() para mantener las hebras en espera.
- Ya en main, se construye el monitor usando un numero de hebras ingresado por el usuario junto con la cantidad de etapas a simular, las etapas se definen con un loop for, que servirá para comprobar la capacidad de reutilización de la barrera.
- En conclusión, el orden de ejecución es:
    - Se construye la barrera y hebras
    - La hebra trabaja usando mutex, si no es la última hebra, entra en espera usando la función incluida en el monitor
    - Cuando la última hebra termina de trabajar, realiza un broadcast para despertar a todas las hebras.
    - La última hebra incrementa la etapa y reinicia el contador a cero, lo que permite la reutilización de la barrera.

Evaluación de resultados:

Se probó el funcionamiento de la barrera con distintas configuraciones de hebras y etapas, en todos los experimentos se mantuvo la funcionalidad esperada, el mutex y las esperas lograron detener las hebras sin problema y la variable de condición fue capaz de despertar a las hebras para continuar de una etapa a la otra.

```
Introduzca el número de hebras: 3
Introduzca el número de etapas: 3

Creando 3 hebras...
Ejecutando 3 etapas

Hebra 0 trabajando en etapa 0
Hebra 1 trabajando en etapa 0
Hebra 2 trabajando en etapa 0
Hebra 0 detenida en etapa 0
Hebra 1 detenida en etapa 0
Hebra 2 detenida en etapa 0
--- Fin de etapa 0 ---

Hebra 2 trabajando en etapa 1
Hebra 0 trabajando en etapa 1
Hebra 1 trabajando en etapa 1
Hebra 1 detenida en etapa 1
Hebra 2 detenida en etapa 1
Hebra 0 detenida en etapa 1
--- Fin de etapa 1 ---

Hebra 0 trabajando en etapa 2
Hebra 1 trabajando en etapa 2
Hebra 2 trabajando en etapa 2
Hebra 1 detenida en etapa 2
Hebra 2 detenida en etapa 2
Hebra 0 detenida en etapa 2
--- Fin de etapa 2 ---


Todas las hebras terminaron después de 3 etapas
```

Parte 2: Simulador de memoria virtual

1. Descripción General

Se desarrollo un simulador virtual con paginación simple y remplazo de reloj (CLOCK). El simulador procesa secuencias de direcciones virtuales **(DV)** de un archivo de traza y decide si cada acceso produce un HIT o un FALLO en la página.

Cuando ocurre el fallo se asigna un marco libre, pero si no quedan marcos disponibles se selecciona un marco víctima y se utilizara el algoritmo CLOCK. Finalmente se calcula el **(DF)** dirección física usando el marco asignado junto con su offset.

2. Arquitectura del simulador y división del trabajo

En la parte 2, de dividió el trabajo en 2 "partes" o "módulos" principales.

- **Modulo A (Matías)**
  Responsable de la traducción de las direcciones virtuales
  - Separación de DV en texto (decimal o hexadecimal):
    - npv = DV >> b
    - offset = DV & (2^b -1)
  - Calculo del valor b, tal que PAGE_SIZE = 2^b.
  - Leer archivos y sus secuencias de trazas.
  - Llamar a procesar_dv() para cada DV

- **Modelo B (Roberto)**
  - implementación completa del algoritmo CLOCK:
    - tabla_paginas[].
    - marcos[].
    - bits de uso.
    - puntero del reloj.
  - Manejo de HIT y FALLO.
  - Construcción de dirección física DF.
  - Implementación del modo –verbose.
  - Contadores de fallos, referencias y tasa de fallos.
  - Función procesar_dv() e imprimir_estadisticas().
  - Integración con el esqueleto del simulador

3. Implementación del algoritmo Clock

El algoritmo **CLOCK** se implementó usando:

- tabla_paginas[npv] a marco
- marcos[marco] a npv
- uso[marco] a bit 0/1
- puntero a índice circular

HIT: si la tabla_paginas[npv] != -1, la página estaría cargada en memoria

FALLO: Si existe un marco libre entonces se asigna directamente, Si no hay marcos libres se ejecuta CLOCK: si uso[puntero] == 0, ese marco es víctima, pero si uso[puntero] == 1, el bit se limpia y se avanza el puntero.
Toda esta logica se desarrolla en las funciones:

- buscar_marco()
- reloj_elegir_victima()
- marcar_uso()

4. Construcción Dirección Física (DF)

Cuando se determina el marco, se usa: “DF = (marco<<b) | offset”. Esto lo que hace es concatenar el marco físico con el desplazamiento dentro del marco.

5. Modo Verbose y estadísticas.

Implementamos un modo opcional, --verbose que muestra cada DV:

- DV original
- npv
- offset
- HIT / FALLO
- marco asignado
- DF calculada

También se logró implementar los contadores:

- total_referencias
- total_fallos
- tasa_fallos = total_fallos / total_referencias

Toda esta lógica se implementó en las funciones:

- Procesar_DV()
- print_verbose()
- imprimir_estadisticas()

6. Ejecución del simulador

El programa acepta: "./sim Nmarcos PAGE_SIZE [--verbose] archivo_traza
Donde:

- Nmarcos: cantidad de marcos físicos disponibles
- PAGE_SIZE: tamaño de marco, debe ser potencia de 2.
- archivo_traza: archivo con dirección virtual por línea.

Ejemplos:

- **./**sim 8 8 trace1.txt
- ./sim 16 4096 –verbose trace2.txt
- ./sim 32 4096 trace.txt

7. Resultados Simulador

Para evaluar el comportamiento del CLOCK, se ejecutó un simulador utilizando 2 trazas: trace1.txt (tamaño de página 8 bytes) y trace2.txt (tamaño de página 4096 bytes). Se midió la tasa de fallos variado con la cantidad de marcos físicos de 8, 16 y 32. Las ejecuciones y los resultados fueron los siguientes:

Trace1.txt (PAGE_SIZE = 8):
./sim 8 8 trace1.txt

```
Marcos = 8 | PageSize = 8 | b = 3 | verbose = 0

--- RESULTADOS ---
Total referencias: 9
Fallos de página: 4
Tasa de fallos: 0.4444
```

./sim 16 4096 trace2.txt

```
Marcos = 16 | PageSize = 4096 | b = 12 | verbose = 0

--- RESULTADOS ---
Total referencias: 7
Fallos de página: 5
Tasa de fallos: 0.7143
```

./sim 32 8 trace1.txt

```
Marcos = 32 | PageSize = 8 | b = 3 | verbose = 0

--- RESULTADOS ---
Total referencias: 9
Fallos de página: 4
Tasa de fallos: 0.4444
```

Trace2.txt(PAGE_SIZE = 4096):

./sim 8 4096 trace2.txt

```
Marcos = 8 | PageSize = 4096 | b = 12 | verbose = 0

--- RESULTADOS ---
Total referencias: 7
Fallos de página: 5
Tasa de fallos: 0.7143
```

./sim 16 4096 trace2.txt

```
Marcos = 16 | PageSize = 4096 | b = 12 | verbose = 0

--- RESULTADOS ---
Total referencias: 7
Fallos de página: 5
Tasa de fallos: 0.7143
```

./sim 32 4096 trace2.txt

```
Marcos = 32 | PageSize = 4096 | b = 12 | verbose = 0

--- RESULTADOS ---
Total referencias: 7
Fallos de página: 5
Tasa de fallos: 0.7143
```

8. Conclusión.

Se logro integrar correctamente la traducción de direcciones virtuales con el manejo del marco y la política de remplazo. El simulador procesa trazas reales, detecta HIT y FALLO, calcula direcciones físicas y entrega estadísticas correctas para los distintos números de marcos. Los resultados obtenidos muestran el comportamiento esperado del algoritmo CLOCK y cumple con los requisitos del enunciado.