Sistemas Operativos: Tarea 2

Profesora: Cecilia Hernandez

Integrantes:

- Benjamin Poblete Castillo
- Carlos Salinas Pereira
- Khristian Villalobos Alfaro

Repositorio: https://github.com/kh12v/tarea-os-memoria-virtual.git
Fecha: 1 de diciembre de 2025

Sincronización con Barreras utilizables

Para la implementación del código de las barreras utilizables se emplearon diferentes componentes como funciones, monitores, barreras, etc. A continuación, se detallan los elementos utilizados:

- ### Barrier_t

  "*Barrier_t*" es un nuevo tipo de dato definido con el propósito de cumplir con la funcionalidad de un monitor, permitiendo encapsular los datos a utilizar. Esto facilita su acceso y mejora la seguridad del código, ya que estos pueden ser modificados sólo desde el propio "*Barrier_t*".

  Variables utilizadas:

  - ☐ **pthread_mutex_t mutex:** Variable mutex, encargada de que no se generen las condiciones de carrera.
  - ☐ **pthread_cond_t cond:** Genera la espera de las hebras que no se están ocupando.
  - ☐ **int count:** Contador de las hebras que se encuentran en ese ciclo de ejecución.
  - ☐ **int N:** Número total de hebras que deberían llegar, es como una condición de salida.
  - ☐ **int stage:** "*stage*" cumple la función de un contador de ciclos, ya que se incrementa solo después de que entre la última hebra a ejecución.

- ## barrier_init

  Esta función se encarga de inicializar todos los datos, asignándoles valores iniciales, como argumento recibe un dato de tipo “*Barrier_t*” y un “int” con el número de Threads. Su ejecución se realiza solo una vez, para que posteriormente se pueda utilizar “*Barrier_t*” sin problemas.

- ## barrier_wait

  La función “*barrier_wait*” tiene la finalidad de asegurarse de que todas las hebras lleguen a esa etapa de la ejecución para poder continuar, lo logra de la siguiente forma:

  - ☐ Toma el mutex para poder modificar variables compartidas con seguridad.
  - ☐ Verifica si es la última hebra en llegar, si es así deja que todas puedan continuar, si no, la hebra dormirá.
  - ☐ Si una hebra es llamada con broadcast, vuelve a verificar si se cumple la condición para continuar.
  - ☐ Se libera el mutex.

- ## barrier_set_N

  Con el uso de un mutex para garantizar que no existan errores lógicos, se define N, qué es la cantidad de hebras requeridas, en caso de no poder capturar el mutex se espera hasta que si se pueda realizar esta modificación con seguridad.

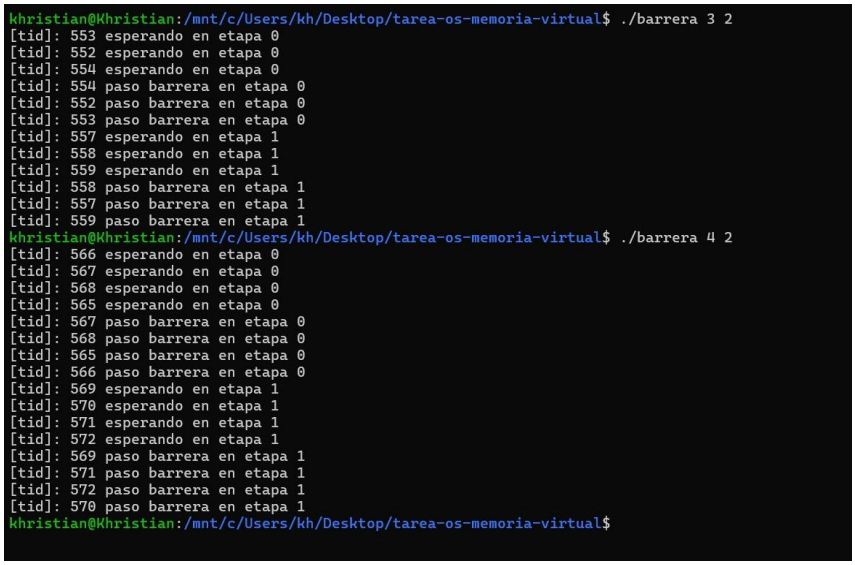
- ## barrier_destroy

  Se limpian y liberan los recursos del sistema cuando ya no se necesita una sincronización.

- ## main

  En el método main se implementa una prueba del mecanismo de sincronización con N hebras que participarán en una cantidad de E ciclos, ambos valores ingresados como argumentos de ejecución. La simulación del uso de las barreras se realiza mediante un ciclo for (para emular las etapas de un código). Una vez finalizado todo el proceso, el programa termina su ejecución y libera los recursos del sistema.

Salida:



La salida muestra los threads, en espera en cada etapa, o también muestra cuando estos pasan.

Simulador de traducción de direcciones

La simulación de traducción de direcciones fue desarrollada de forma que se utiliza como entradas los valores almacenados en dos archivos de texto diferentes: "*trace1.txt*" y "*trace2.txt*", los cuales contienen las direcciones de memoria en formato hexadecimal.

- El programa realiza validaciones de la siguiente manera:

```
./sim <Nmarcos> <tamañomarco> [--verbose] <traza.txt>
```

Donde:

- ☐ **Nmarcos:** Número de marcos (int)
- ☐ **tamañomarco:** Tamaño de cada marco (int)
- ☐ **–verbose:** Parámetro opcional, si se utiliza se muestra información sobre el proceso de traducción
- ☐ **traza.txt:** El archivo del cual se va a obtener el input del programa (cada línea es una dirección virtual en hexadecimal)

- Para realizar la simulación de un marco de página, se utiliza el siguiente struct

```
typedef struct {
    int pagina_virtual;
```

```
    int ocupado;
    int uso;
} Marco;
```

Todo marco porta distintos valores iniciales, con las siguientes finalidades:

- ☐ **pagina_virtual:** Inicialmente tiene el valor -1, se utiliza para indicar que página del proceso está guardada ahí.
- ☐ **ocupado:** Inicialmente tiene el valor 0, indica si el marco es válido o está vacío.
- ☐ **uso:** Inicialmente tiene el valor 0, representa el bit de referencia, es utilizado por el algoritmo de reloj, si el valor es 1 significa que fue referenciado recientemente y se le da otra oportunidad para no ser reemplazado.

- El primer paso para la traducción de una dirección virtual consta de separar el número de página virtual (NPV) y offset.

```
unsigned MASK = (1U << b) - 1;
```

```
dv = (unsigned int)strtol(buffer, NULL, 0);
offset = dv & MASK;
nvp = dv >> b;
```

El valor del offset se obtiene aplicando una máscara (MASK), después el número de página virtual (NPV) se calcula desplazando la dirección virtual hacia la derecha según la cantidad de bits correspondiente al *offset*, determinada por el tamaño del marco.

Pasos de Simulación

1) Búsqueda de el valor de NVP:

Se recorre el array “marcos” en busca del valor NVP correspondiente a la dirección virtual. En caso de que se encuentre (HIT), calcula la dirección física, se cambia el valor de “*uso*” del marco como 1 y posterior a esto se continúa con la siguiente dirección de memoria.

2) Caso MISS:

En caso de no encontrar el valor NVP en los marcos, se cargará un marco nuevo. Primero se busca un marco libre, es decir, si el valor de "*ocupado*" de algún marco es cero este se utilizara, en caso que no exista alguno disponible se activa el algoritmo de reloj.

3) Algoritmo de reloj:

La funcionalidad del algoritmo es a través del puntero "*reloj*". Se señala un marco dentro del arreglo "marcos" existentes, a continuación esta señalización va circulando como una aguja de reloj dentro del arreglo, es decir, al llegar al final, la continuación será el valor inicial. Si el marco que señala el puntero tiene un valor de "*uso*" igual a 1, se cambia por un 0, luego el puntero avanza al siguiente elemento. Cuando el puntero se encuentra con un marco cuyo valor de "*uso*" es igual a 0, entonces se cambia la página antigua por la nueva, después se actualizan los datos del marco y el puntero pasa al siguiente elemento.

4) Valores estadísticos:

Al finalizar la ejecución del programa, Se muestran los siguientes datos

- Número de páginas procesadas
- Total de fallos de página (MISS)
- Tasa de fallos (se determina como el porcentaje entre la relación [fallos de página] / [páginas procesadas]).

Resultados

- Prueba 1: **./sim 16 256 trace1.txt**

Estadísticas:
Total de fallos de página: 16
Total de referencias de página: 8192
Tasa de fallos de página: 0.20%

- Prueba 2: **./sim 4 1024 –verbose trace2.txt**

Estadísticas:
Total de fallos de página: 8120
Total de referencias de página: 8192
Tasa de fallos de página: 99.12%