

Concurrencia:

La concurrencia de procesos es un concepto clave en el diseño de sistemas operativos, ya que permite que múltiples procesos se ejecuten de manera aparentemente simultánea. Esto mejora la eficiencia y el rendimiento del sistema al aprovechar mejor los recursos disponibles

Problemas y Soluciones en la Concurrencia

- **Sincronización:** asegura que los hilos o procesos que comparten recursos no interfieran entre sí de manera destructiva. Se utilizan primitivas como semáforos, mutexes y monitores para este propósito.
- **Condiciones de carrera (Race Conditions):** ocurren cuando múltiples procesos o hilos acceden y manipulan datos compartidos y el resultado depende del orden de ejecución. Se evitan usando mecanismos de sincronización.
- **Sección crítica (Critical Section):** parte del código donde se accede a recursos compartidos. Se debe proteger con mecanismos de sincronización para evitar condiciones de carrera.
- **Deadlock (Interbloqueo):** situación donde dos o más procesos quedan bloqueados esperando recursos que los otros procesos tienen. Para evitar deadlocks, se pueden usar algoritmos de prevención, detección y recuperación.
- **Starvation (Hambre):** ocurre cuando un proceso no obtiene los recursos necesarios para su ejecución debido a que otros procesos los están acaparando. Se mitiga mediante técnicas de programación justa (fair scheduling).

Concurrencia:

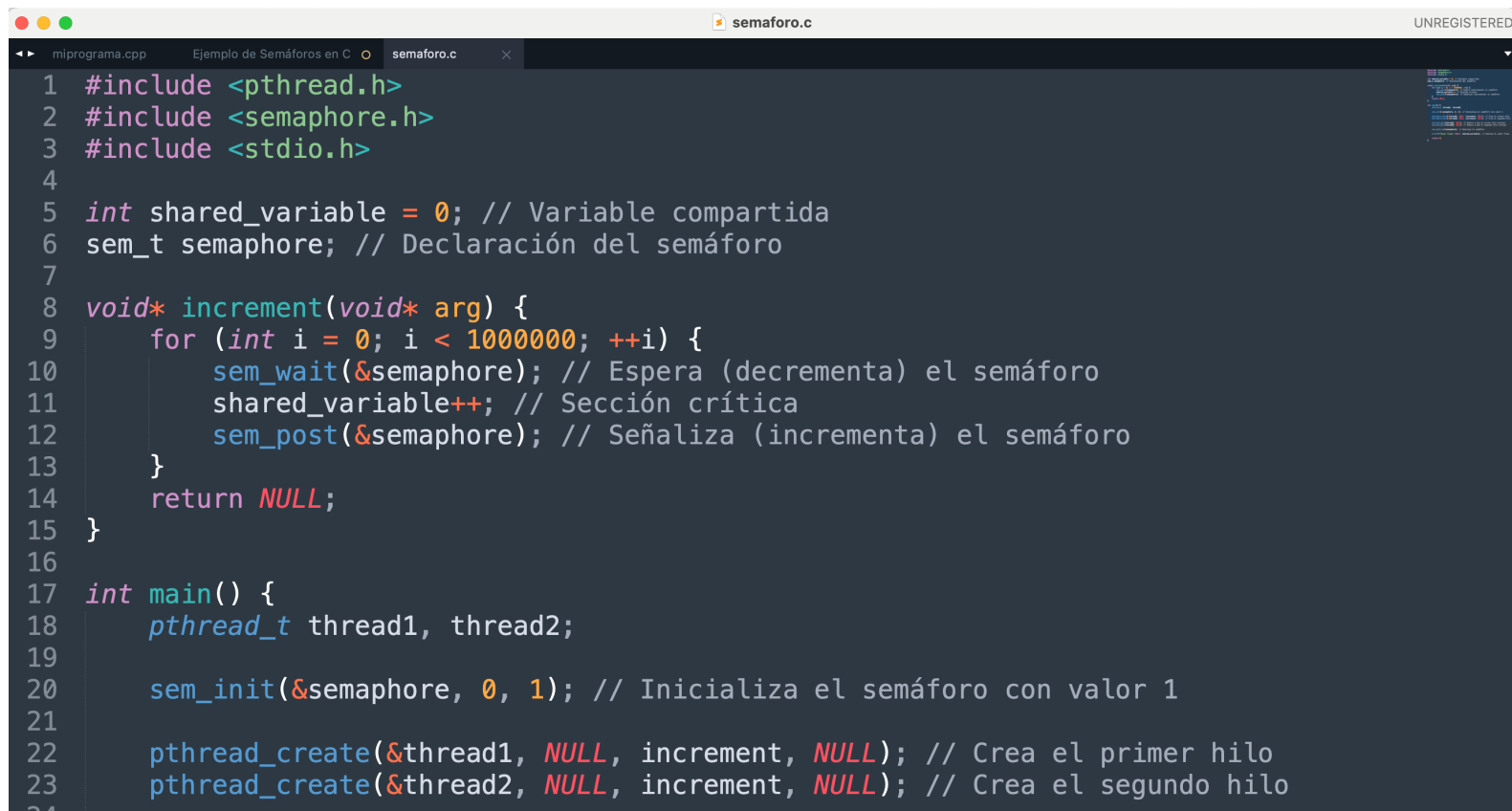
Algoritmos y Herramientas

- 1.Semáforos:** variables utilizadas para controlar el acceso a recursos compartidos. Pueden ser binarios (0 o 1) o contadores.
- 2.Mutexes:** abreviatura de "Mutual Exclusion", es un tipo de semáforo utilizado para asegurar que solo un hilo acceda a una sección crítica a la vez.
- 3.Monitores:** estructuras de alto nivel que proporcionan una forma más abstracta de manejar la sincronización.
- 4.Algoritmos de planificación (Scheduling):** determinan el orden en que los procesos o hilos se ejecutan. Ejemplos incluyen Round Robin, FIFO, y planificación basada en prioridades.

Ejemplos:

Semáforos

Este ejemplo muestra dos hilos que incrementan una variable compartida, usando un semáforo para asegurar que solo un hilo a la vez pueda modificar la variable.



```
1 #include <pthread.h>
2 #include <semaphore.h>
3 #include <stdio.h>
4
5 int shared_variable = 0; // Variable compartida
6 sem_t semaphore; // Declaración del semáforo
7
8 void* increment(void* arg) {
9     for (int i = 0; i < 1000000; ++i) {
10         sem_wait(&semaphore); // Espera (decrementa) el semáforo
11         shared_variable++; // Sección crítica
12         sem_post(&semaphore); // Señaliza (incrementa) el semáforo
13     }
14     return NULL;
15 }
16
17 int main() {
18     pthread_t thread1, thread2;
19
20     sem_init(&semaphore, 0, 1); // Inicializa el semáforo con valor 1
21
22     pthread_create(&thread1, NULL, increment, NULL); // Crea el primer hilo
23     pthread_create(&thread2, NULL, increment, NULL); // Crea el segundo hilo
24 }
```

Ejemplos:

Mutexes

Este ejemplo muestra dos hilos que incrementan una variable compartida, utilizando un mutex para asegurar que solo un hilo a la vez pueda modificar la variable.

¿Cómo Funciona el Mutex?

- **Bloqueo (Locking):** `pthread_mutex_lock(&mutex)` intenta adquirir el bloqueo del mutex. Si el mutex ya está bloqueado por otro hilo, el hilo actual se bloquea hasta que el mutex esté disponible.
- **Desbloqueo (Unlocking):** `pthread_mutex_unlock(&mutex)` libera el bloqueo del mutex, permitiendo que otros hilos puedan adquirirlo.

El uso de mutexes asegura que solo un hilo a la vez pueda ejecutar la sección crítica del código, evitando condiciones de carrera y asegurando la integridad de los datos compartidos.

Ejemplos:

Monitores

Los monitores son una abstracción de alto nivel utilizada para manejar la sincronización en sistemas concurrentes. Un monitor encapsula variables compartidas, procedimientos y los mecanismos de sincronización, garantizando que solo un hilo puede ejecutar un procedimiento del monitor a la vez.

Ejemplos:

Algoritmo de Planificación Round Robin

El algoritmo Round Robin asigna un tiempo fijo (quantum) a cada proceso en la cola de procesos listos. Cada proceso se ejecuta durante un quantum de tiempo y luego se coloca al final de la cola si no ha terminado su ejecución.

Introduzca el número de procesos: 3

Introduzca el tiempo de llegada del proceso 1: 0

Introduzca el tiempo de ráfaga para el proceso 1: 10

Tiempo de llegada del proceso 2: 1

Introduzca el tiempo de ráfaga para el proceso 2: 5

Introducir el tiempo de llegada para el proceso 3: 2

Introducir el tiempo de ráfaga para el proceso 3: 8

Introducir el cuanto de tiempo: 3

Ejemplos:

Algoritmo de Planificación Round Robin

```
4_RR — -zsh — 109x29
(base) joaquinfsanchez@MacBook-Air-de-Joaquin 4_RR % ls
RR.c
(base) joaquinfsanchez@MacBook-Air-de-Joaquin 4_RR % cc RR.c
(base) joaquinfsanchez@MacBook-Air-de-Joaquin 4_RR % ./a.out
Introduzca el número de procesos: 3
Introduzca la hora de llegada del proceso 1: 0
Introduzca el tiempo de ráfaga para el proceso 1: 10
Introduzca la hora de llegada del proceso 2: 1
Introduzca el tiempo de ráfaga para el proceso 2: 5
Introduzca la hora de llegada del proceso 3: 2
Introduzca el tiempo de ráfaga para el proceso 3: 8
Introduzca el cuanto de tiempo: 3
PID      Time de llegada Timepo de Rafaga      Timepo Competicion      Turnaround Time Timepo Espera
1         0             10             23             23             13
2         1              5             14             13              8
3         2              8             22             20             12
(base) joaquinfsanchez@MacBook-Air-de-Joaquin 4_RR %
```

Ejemplos:

Algoritmo de Planificación Round Robin

Explicación del Ejemplo de Salida:

- El proceso 1 llega primero y se ejecuta durante el quantum de 3 unidades de tiempo, luego se coloca al final de la cola.
- El proceso 2 llega y se ejecuta durante el quantum de 3 unidades de tiempo, luego se coloca al final de la cola.
- El proceso 3 llega y se ejecuta durante el quantum de 3 unidades de tiempo, luego se coloca al final de la cola.
- El proceso 1 se ejecuta de nuevo, y así sucesivamente hasta que todos los procesos terminen.