**PRÁCTICA 2 DE SISTEMAS OPERATIVOS**

**TEMA: Creación de Hilos**

**Nombres:** Jeremy Jiménez y Fernando Huilca

**Carrera:** Ingeniería de Software

**Grupo:** GR1SW

**Fecha:** 22 / 07 / 2024

**Índice de Contenidos**

[1. OBJETIVOS 2](#_Toc172571049)

[2. INFORME 2](#_Toc172571050)

[Código sin semáforos 2](#_Toc172571051)

[Código con semáforos 3](#_Toc172571052)

[Código con 3 hilos e impresión de semáforo 4](#_Toc172571053)

[Código con semáforo. Variable color de texto. 6](#_Toc172571054)

[Código con mutex. 7](#_Toc172571055)

[3. CONCLUSIONES Y RECOMENDACIONES 9](#_Toc172571056)

[4. BIBLIOGRAFÍA 10](#_Toc172571057)

**Índice de Imágenes**

[Ilustración 1 Ejecución de código semáforo. 2](#_Toc172570530)

[Ilustración 2 Ejecución código con semáforo. 3](#_Toc172570531)

[Ilustración 3 Ejecución de semáforos con 3 hilos. 5](#_Toc172570532)

[Ilustración 4 Código con semáforo. Variable color de texto. 7](#_Toc172570533)

[Ilustración 5 Código con mutex. 9](#_Toc172570534)

# OBJETIVOS

1.1. Implementar el uso de semáforos y mutex en C.

1.2. Asimilar los conceptos teóricos revisados en clase.

# INFORME

## Código sin semáforos

* Ejecutar varias ocasiones el código sin semáforos con la variable MAX igual a 1000, 10000, 100000, 1000000

|  |  |
| --- | --- |
| **MAX** | **a** |
| 1000 | 0 |
| 10000 | 0 |
| 100000 | 0 |
| 1000000 | 0 |

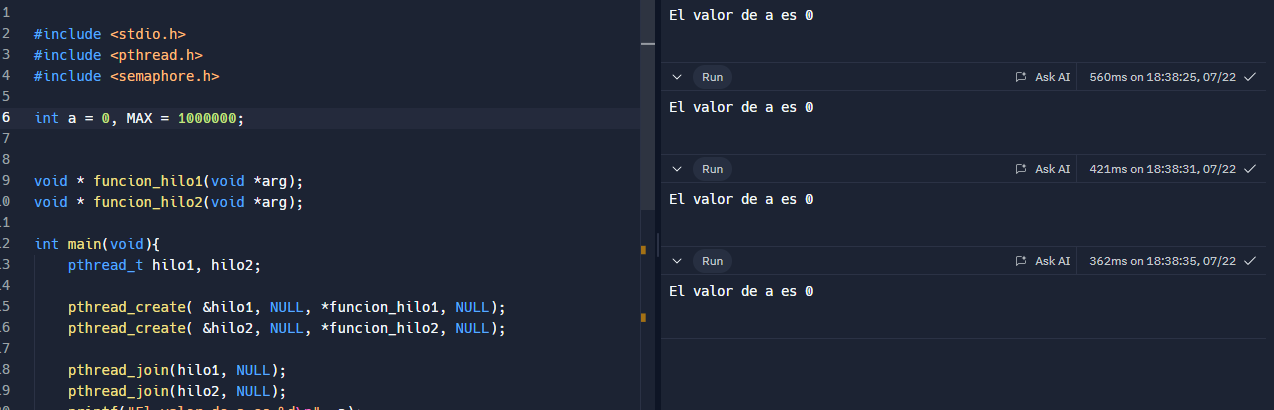


Ilustración 1 Ejecución de código semáforo.

Independientemente del valor que tome la variable MAX, a siempre va a imprimir en total “0”, ya que el primer hilo incrementa a la variable ‘a’ MAX veces, y, el siguiente hilo, decrementa a ‘a’ MAX veces. MAX es igual en todos los casos por ende se anula o su resultado es igual a 0.

## Código con semáforos

* Ejecutar el código con semáforos con la variable MAX igual a 1000000, 1e9, 1e12.

|  |  |
| --- | --- |
| **MAX** | **a** |
| 1000000 | 0 |
| 1e9 | 0 |
| 1e12 | 0 |

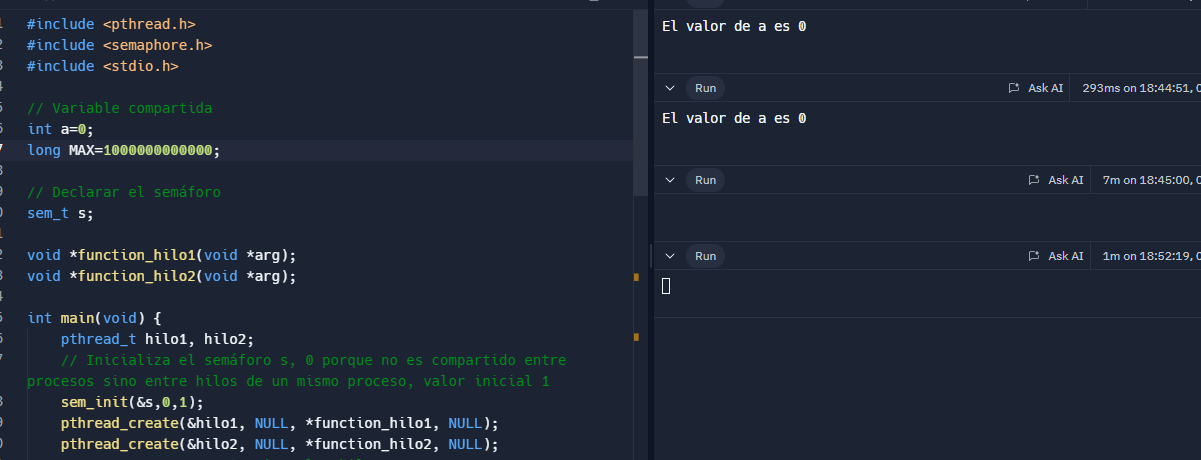


Ilustración 2 Ejecución código con semáforo.

Al igual que en la ejecución anterior, el valor de ‘a’ es igual a 0, la variable MAX incrementa y disminuye el valor de ‘a’ en cada ejecución de hilos, por ende, al final termina anulándose.

## Código con 3 hilos e impresión de semáforo

* **Para el código de la Figura 2, añadir un tercer hilo e imprimir en pantalla el valor del semáforo durante cada ejecución.**
* **Colocar el código propuesto en el informe con comentarios en las instrucciones más relevantes.**
* **Código de ejecutado:**

**#include <pthread.h>**

**#include <semaphore.h>**

**#include <stdio.h>**

**// Variable compartida**

**int a = 0;**

**long MAX = 1000000000;**

**// Declarar el semáforo**

**sem\_t s;**

**void \*function\_hilo1(void \*arg);**

**void \*function\_hilo2(void \*arg);**

**void \*function\_hilo3(void \*arg);**

**int main(void) {**

**pthread\_t hilo1, hilo2, hilo3;**

**// Inicializa el semáforo s, 0 porque es compartido entre hilos del mismo proceso, valor inicial 1**

**sem\_init(&s, 0, 1);**

**pthread\_create(&hilo1, NULL, function\_hilo1, NULL);**

**pthread\_create(&hilo2, NULL, function\_hilo2, NULL);**

**pthread\_create(&hilo3, NULL, function\_hilo3, NULL);**

**// Esperar que terminen los hilos**

**pthread\_join(hilo1, NULL);**

**pthread\_join(hilo2, NULL);**

**pthread\_join(hilo3, NULL);**

**printf("El valor final de a es %d\n", a);**

**// Destruir el semáforo**

**sem\_destroy(&s);**

**return 0;**

**}**

**void \*function\_hilo1(void \*arg) {**

**for (int i = 0; i < MAX; i++) {**

**sem\_wait(&s); // Bloquea la variable compartida**

**a++;**

**sem\_post(&s); // Incrementa el valor del semáforo**

**}**

**}**

**void \*function\_hilo2(void \*arg) {**

**for (int i = 0; i < MAX; i++) {**

**sem\_wait(&s);**

**a--;**

**sem\_post(&s);**

**}**

**}**

**void \*function\_hilo3(void \*arg) {**

**//for (int i = 0; i < MAX; i++) {**

**sem\_wait(&s);**

**// Realiza alguna operación (ejemplo: imprime un mensaje)**

**int sem\_value;**

**sem\_getvalue(&s, &sem\_value);**

**printf("Hilo 3: Semáforo = %d\n", sem\_value);**

**sem\_post(&s);**

**//}**

**}**

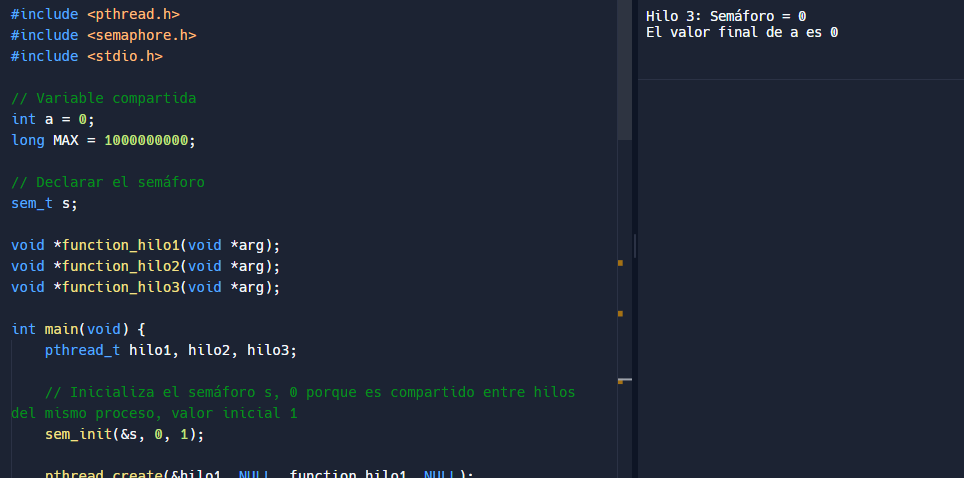


Ilustración 3 Ejecución de semáforos con 3 hilos.

## Código con semáforo. Variable color de texto.

**Tomar como base el código de la Figura 3 propuesto y añadir un semáforo**

**Código a ejecutarse:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <pthread.h>**

**#include <semaphore.h>**

**sem\_t semaforo;**

**void \*rojo(void \*id)**

**{**

**sem\_wait(&semaforo);**

**#define A "\x1b[31m"**

**printf(A "Este texto es ROJO! \n");**

**sem\_post(&semaforo);**

**return NULL;**

**}**

**void \*verde(void \*id)**

**{**

**sem\_wait(&semaforo);**

**#define B "\x1b[32m"**

**printf(B "Este texto es VERDE! \n");**

**sem\_post(&semaforo);**

**return NULL;**

**}**

**int main()**

**{**

**pthread\_t hilo\_rojo, hilo\_verde;**

**sem\_init(&semaforo, 0, 1);**

**pthread\_create(&hilo\_rojo, NULL, rojo, NULL);**

**pthread\_create(&hilo\_verde, NULL, verde, NULL);**

**pthread\_join(hilo\_rojo, NULL);**

**pthread\_join(hilo\_verde, NULL);**

**#define C "\x1b[34m"**

**printf(C "Este texto es AZUL \n");**

**sem\_destroy(&semaforo);**

**return 0;**

**}**

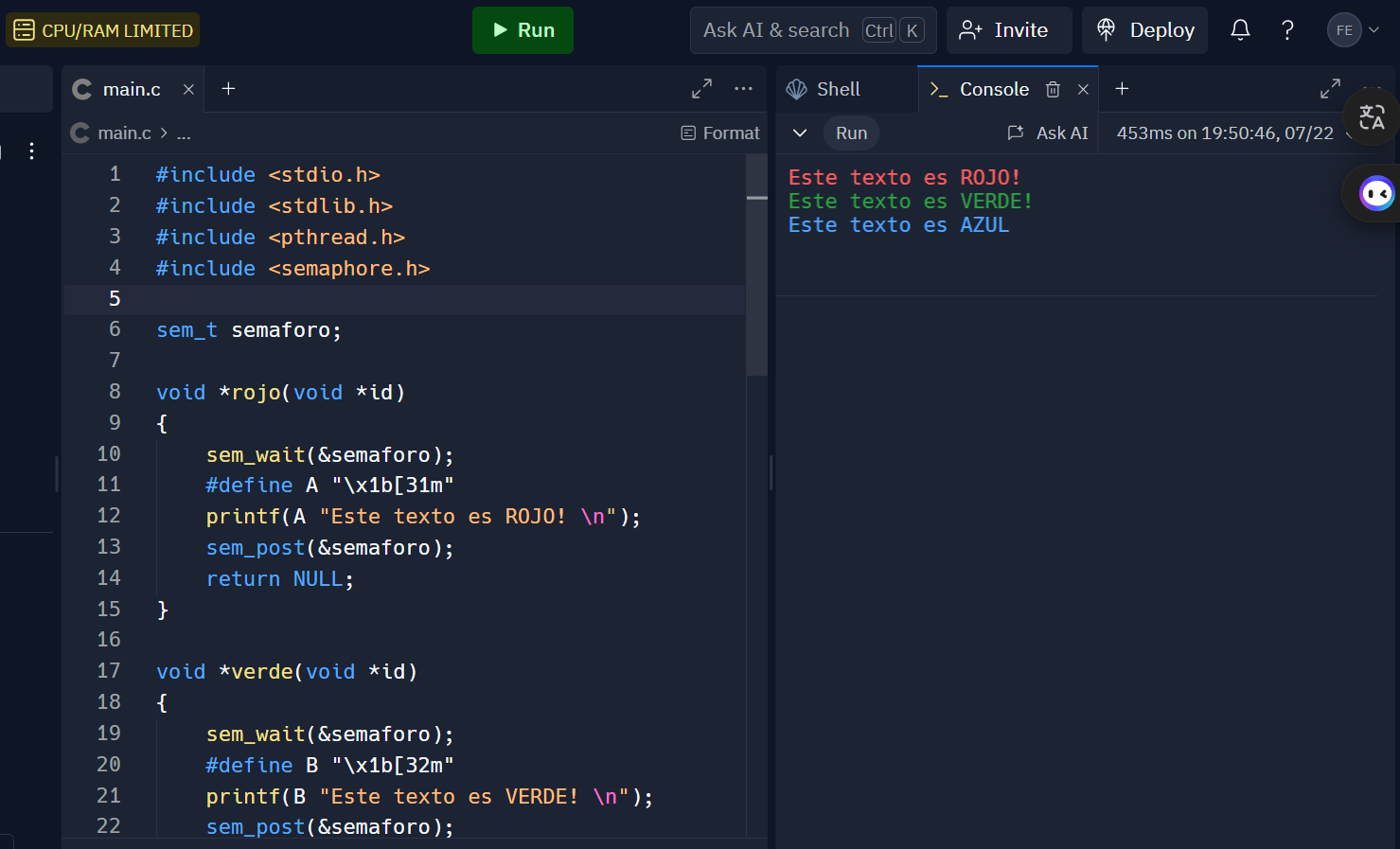
****

Ilustración 4 Código con semáforo. Variable color de texto.

## Código con mutex.

**Tomar como base los códigos de las Figuras 3 y 4 y verificar que un mutex permita compartir la variable que define el color del texto.**

**Código a ejecutarse:**

**#include <pthread.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#define MAX 100000000**

**pthread\_mutex\_t mutex;**

**void \*rojo(void \*id) {**

**pthread\_mutex\_lock(&mutex);**

**#define A "\x1b[31m"**

**printf(A "Este texto es ROJO! \n");**

**pthread\_mutex\_unlock(&mutex);**

**return NULL;**

**}**

**void \*verde(void \*id) {**

**pthread\_mutex\_lock(&mutex);**

**#define B "\x1b[32m"**

**printf(B "Este texto es VERDE! \n");**

**pthread\_mutex\_unlock(&mutex);**

**return NULL;**

**}**

**void \*funcion\_hilo1(void \*arg) {**

**pthread\_mutex\_t \*mutex = arg;**

**pthread\_mutex\_lock(mutex);**

**for (int i = 0; i < MAX; i++) {**

**// Acción del hilo 1**

**}**

**pthread\_mutex\_unlock(mutex);**

**return NULL;**

**}**

**void \*funcion\_hilo2(void \*arg) {**

**pthread\_mutex\_t \*mutex = arg;**

**pthread\_mutex\_lock(mutex);**

**for (int i = 0; i < MAX; i++) {**

**// Acción del hilo 2**

**}**

**pthread\_mutex\_unlock(mutex);**

**return NULL;**

**}**

**int main() {**

**pthread\_t hilo1, hilo2, hilo\_rojo, hilo\_verde;**

**pthread\_mutex\_init(&mutex, NULL);**

**pthread\_create(&hilo\_rojo, NULL, rojo, NULL);**

**pthread\_create(&hilo\_verde, NULL, verde, NULL);**

**pthread\_join(hilo\_rojo, NULL);**

**pthread\_join(hilo\_verde, NULL);**

**pthread\_create(&hilo1, NULL, funcion\_hilo1, &mutex);**

**pthread\_create(&hilo2, NULL, funcion\_hilo2, &mutex);**

**pthread\_join(hilo1, NULL);**

**pthread\_join(hilo2, NULL);**

**#define C "\x1b[34m"**

**printf(C "Este texto es AZUL \n");**

**pthread\_mutex\_destroy(&mutex);**

**return 0;**

**}**

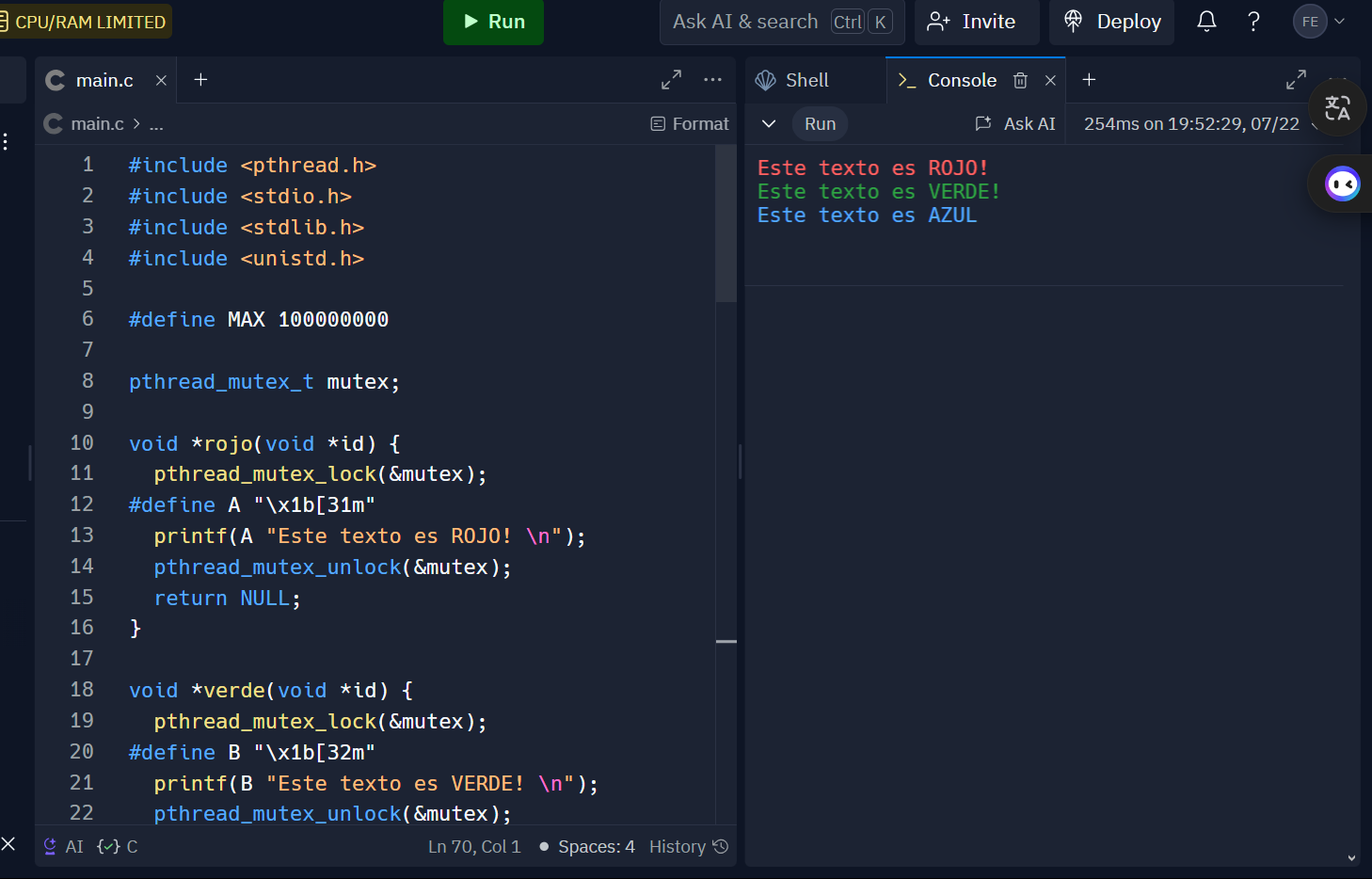
****

Ilustración 5 Código con mutex.

# CONCLUSIONES Y RECOMENDACIONES

Conclusiones:

* La implementación de semáforos y mutex permite una sincronización eficiente entre hilos, asegurando que los recursos compartidos se manejen de manera adecuada y evitando condiciones de carrera.
* En el caso de los semáforos, su uso es crucial cuando se necesita un mecanismo de espera activa que permita a los hilos acceder a recursos limitados de manera controlada.

Recomendaciones:

* Siempre documentar y comentar el código, especialmente en las secciones donde se implementan mecanismos de sincronización, para facilitar la comprensión y mantenimiento del código por parte de otros desarrolladores.
* Realizar pruebas exhaustivas bajo diferentes escenarios y cargas para asegurar que los mecanismos de sincronización están funcionando correctamente y no introducen nuevos problemas como deadlocks o starvation.

# BIBLIOGRAFÍA

[1] Imagine Apps, "Servicio de software Flutter Flow," Imagine Apps. [Online]. Available: https://www.imagineapps.co/servicio-de-software/flutter-flow. [Accessed: May 26, 2024].

[2] G. Lawrence, "FlutterFlow Basics: Building Your First App," YouTube. [Online]. Available: https://www.youtube.com/watch?v=hLoVTSAf4tA. [Accessed: May 26, 2024].

[3] Intel, "Intel Core i7 Processors," Intel. [Online]. Available: https://www.intel.la/content/www/xl/es/products/details/processors/core/i7/products.html. [Accessed: May 26, 2024].

[4] Intel, "Procesador Intel® Core™ i5-3470," Intel. [Online]. Available: https://www.intel.la/content/www/xl/es/products/sku/68316/intel-core-i53470-processor-6m-cache-up-to-3-60-ghz/specifications.html. [Accessed: May 26, 2024].