

## 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
2. INFORME.....	2
Código sin semáforos .....	2
Código con semáforos .....	3
Código con 3 hilos e impresión de semáforo .....	4
Código con semáforo. Variable color de texto.....	6
Código con mutex.....	7
3. CONCLUSIONES Y RECOMENDACIONES.....	9
4. BIBLIOGRAFÍA .....	10

### Índice de Imágenes

Ilustración 1 Ejecución de código semáforo.....	2
Ilustración 2 Ejecución código con semáforo.....	3
Ilustración 3 Ejecución de semáforos con 3 hilos. ....	5
Ilustración 4 Código con semáforo. Variable color de texto. ....	7
Ilustración 5 Código con mutex. ....	9

## 1. OBJETIVOS

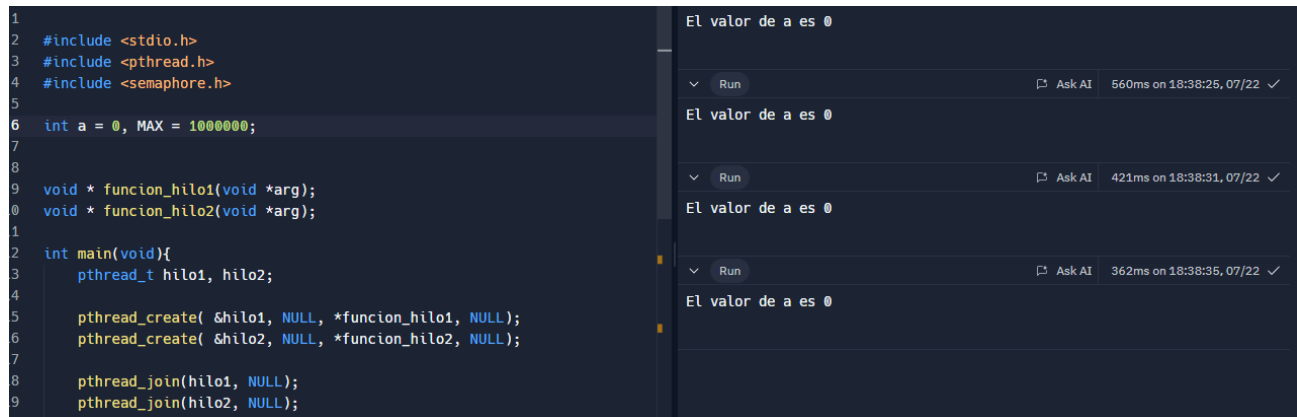
- 1.1. Implementar el uso de semáforos y mutex en C.
- 1.2. Asimilar los conceptos teóricos revisados en clase.

## 2. 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



```
1
2 #include <stdio.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5
6 int a = 0, MAX = 1000000;
7
8 void * funcion_hilo1(void *arg);
9 void * funcion_hilo2(void *arg);
10
11 int main(void){
12     pthread_t hilo1, hilo2;
13
14     pthread_create( &hilo1, NULL, *funcion_hilo1, NULL);
15     pthread_create( &hilo2, NULL, *funcion_hilo2, NULL);
16
17     pthread_join(hilo1, NULL);
18     pthread_join(hilo2, NULL);
19     printf("El valor de a es %d\n", a);
20 }
```

El valor de a es 0

Run Ask AI 560ms on 18:38:25, 07/22 ✓

El valor de a es 0

Run Ask AI 421ms on 18:38:31, 07/22 ✓

El valor de a es 0

Run Ask AI 362ms on 18:38:35, 07/22 ✓

El valor de a es 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

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

// Variable compartida
int a=0;
long MAX=1000000000000;

// Declarar el semáforo
sem_t s;

void *function_hilo1(void *arg);
void *function_hilo2(void *arg);

int main(void) {
    pthread_t hilo1, hilo2;
    // Inicializa el semáforo s, 0 porque no es compartido entre
    // procesos sino entre hilos de un mismo proceso, valor inicial 1
    sem_init(&s,0,1);
    pthread_create(&hilo1, NULL, *function_hilo1, NULL);
    pthread_create(&hilo2, NULL, *function_hilo2, NULL);
}
```

El valor de a es 0

Run Ask AI 293ms on 18:44:51,0

El valor de a es 0

Run Ask AI 7m on 18:45:00,0

Run Ask AI 1m on 18:52:19,0

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);
    //}
}
```

```
#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);
```

Hilo 3: Semáforo = 0  
El valor final de a es 0

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);

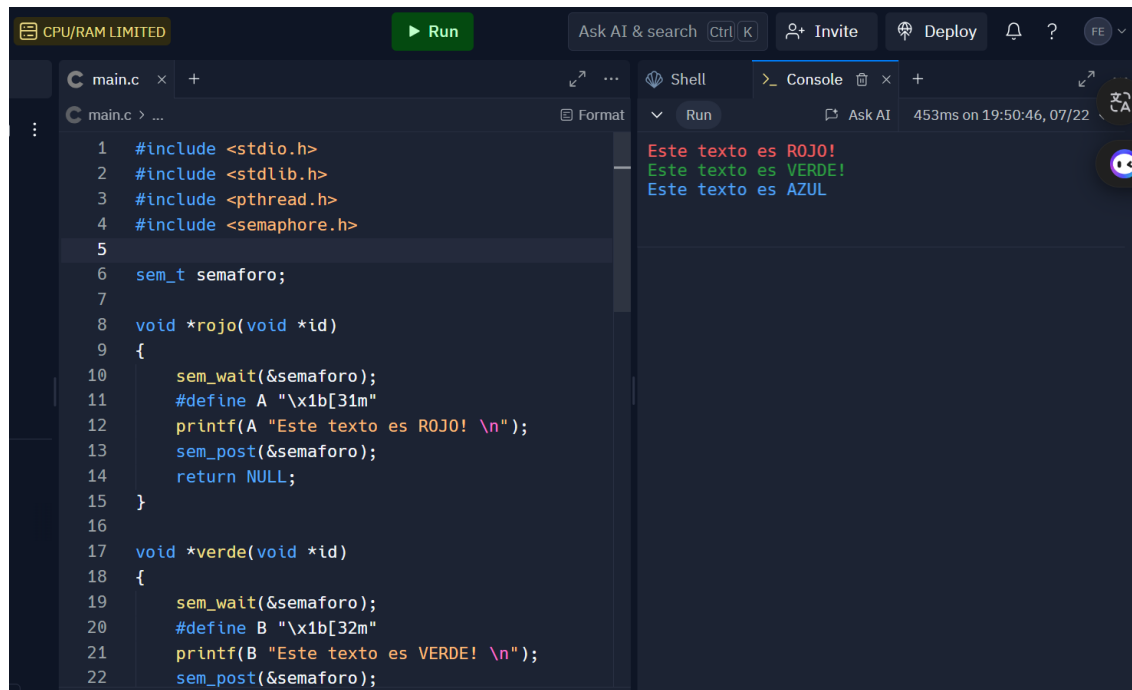
    pthread_create(&hilo_rojo, NULL, rojo, NULL);
    pthread_create(&hilo_verde, NULL, verde, NULL);

    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;
}
```



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5
6 sem_t semaforo;
7
8 void *rojo(void *id)
9 {
10     sem_wait(&semaforo);
11     #define A "\x1b[31m"
12     printf(A "Este texto es ROJO! \n");
13     sem_post(&semaforo);
14     return NULL;
15 }
16
17 void *verde(void *id)
18 {
19     sem_wait(&semaforo);
20     #define B "\x1b[32m"
21     printf(B "Este texto es VERDE! \n");
22     sem_post(&semaforo);
23 }
```

Console output:

```
Este texto es ROJO!
Este texto es VERDE!
Este texto es AZUL
```

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);

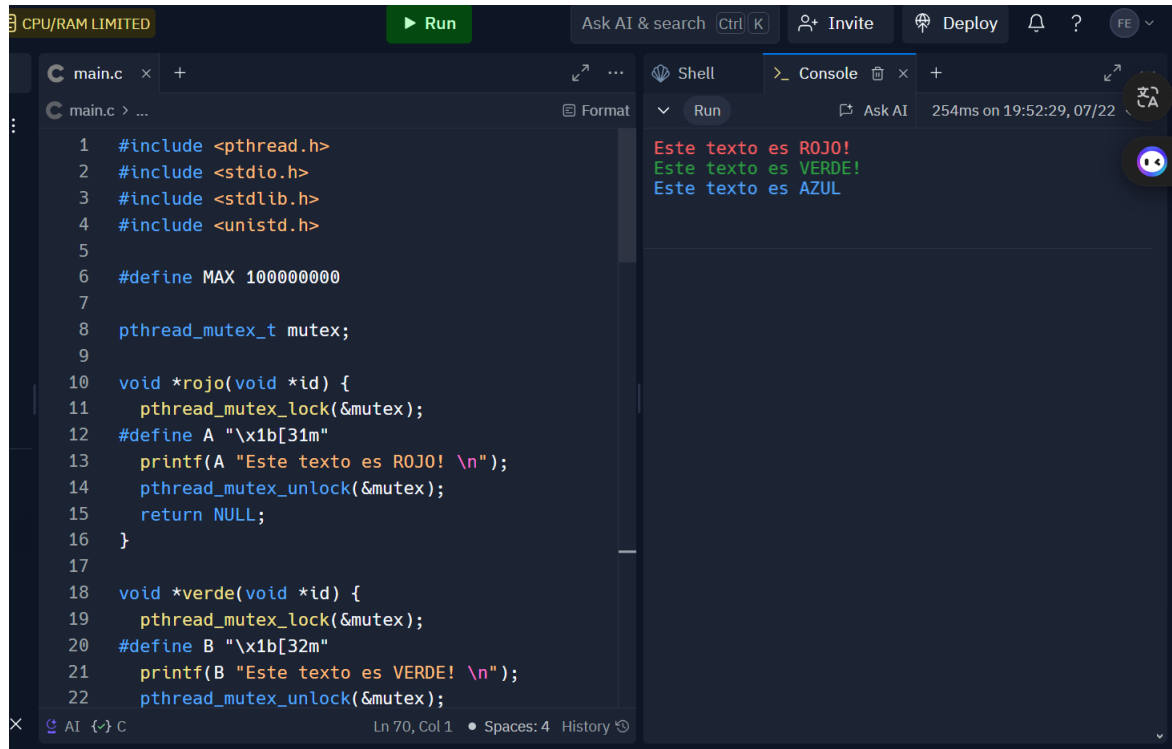
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);

#define C "\x1b[34m"
    printf(C "Este texto es AZUL \n");

    pthread_mutex_destroy(&mutex);

    return 0;
}
```





```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5
6 #define MAX 1000000000
7
8 pthread_mutex_t mutex;
9
10 void *rojo(void *id) {
11     pthread_mutex_lock(&mutex);
12     #define A "\x1b[31m"
13     printf(A "Este texto es ROJO! \n");
14     pthread_mutex_unlock(&mutex);
15     return NULL;
16 }
17
18 void *verde(void *id) {
19     pthread_mutex_lock(&mutex);
20     #define B "\x1b[32m"
21     printf(B "Este texto es VERDE! \n");
22     pthread_mutex_unlock(&mutex);
23 }
```

Este texto es ROJO!  
Este texto es VERDE!  
Este texto es AZUL

Ilustración 5 Código con mutex.

### 3. 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.

---

## 4. 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].