LABORATORIO DE SISTEMAS OPERATIVOS

PERÍODO ACADÉMICO: 2025 – B **EQUIPO:**

PROFESOR: Ing. Marcela Saavedra MSc.

TIPO DE INSTRUMENTO: Guía de Laboratorio

TEMA: USO DE SEMÁFOROS

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1. OBJETIVOS

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

2. MARCO TEÓRICO

Semáforo

Mecanismo de sincronización que protege una sección crítica.

El semáforo es un tipo de datos abstracto o variable que se utiliza para controlar el acceso a un recurso común mediante múltiples procesos en un sistema concurrente, como un sistema operativo multitarea.

Mutex

En la programación concurrente, Mutex es un objeto en un programa que sirve como bloqueo, usado para negociar la exclusión mutua entre subprocesos. Mutex es un caso especial del semáforo; es un objeto de exclusión mutua que sincroniza el acceso a un recurso.

Funciones utilizadas en C

```
//Inicializa el semáforo
        int sem_init(sem_t *sem, int pshared, unsigned int value);

//Decrementa el valor del semáforo
        int sem_wait (sem_t *sem);

//Incrementa el valor del semáforo, sem_signal
        int sem_post(sem_t *sem);
```

Librería a utilizar

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

3. PROCEDIMIENTO

3.1 Código sin semáforos

Ejecutar varias ocasiones el código Sin semáforos con la variable MAX igual a 1000, 10000, 100000, 1000000.
Completar la tabla 1 con el valor de la variable a que se imprime en pantalla.
Comentar por qué se obtienen los resultados.

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


//Variable compartida
int a=0,MAX=10;

void * funcion_hilo1(void *arg);
void * funcion_hilo2(void *arg);

int main (void)
{
        pthread_t hilo1, hilo2;
        pthread_create(&hilo1, NULL, *funcion_hilo1, NULL);
        pthread_create(&hilo2, NULL, *funcion_hilo2, NULL);
        //Para esperar que terminen los hilos
        pthread_join(hilo1, NULL);
        pthread_join(hilo2, NULL);
        printf ("El valor de a es %d",a);
        return 0;
}

void * funcion_hilo1(void *arg)
{
        for(int i=0; i<MAX; i++)
        {
                a +=1;
        }
}

void * funcion_hilo2(void *arg)
{
        for(int i=0; i<MAX; i++)
        {
                a -=1;
        }
}
```

Figura 1. Código sin semáforos

Para ejecutar utilizamos: gcc -pthread figura1.c -std=c99 -o figura1

felipemerino@localhost:~ — vim ej3_1.c

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

// Variable compartida
int a = 0;
// variable: 1000, 10000, 100000, 1000000
long MAX = 10000;

void * funcion_hilo1(void *arg);
void * funcion_hilo2(void *arg);

int main (void) {
    pthread_t hilo1, hilo2;

    // Creación de hilos
    pthread_create(&hilo1, NULL, funcion_hilo1, NULL);
    pthread_create(&hilo2, NULL, funcion_hilo2, NULL);

    // Esperar a que terminen los hilos
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);

    printf("El valor de a es %d \n", a);
    return 0;
}

void * funcion_hilo1(void *arg) {
    for(int i=0; i<MAX; i++) {
        a += 1;
    }
    return NULL;
}

void * funcion_hilo2(void *arg) {
    for(int i=0; i<MAX; i++) {
        a -= 1;
    }
    return NULL;
}
```

INSERT ej3_1.c[+] c 17% ln : 7 %:17

-- INSERTAR --

```
[felipemerino@localhost ~]$ vim ej3_1.c
[felipemerino@localhost ~]$ gcc -pthread ej3_1.c -std=c99 -o ej3_1
[felipemerino@localhost ~]$ ./ej3_1
El valor de a es 9212
```

Tabla1.

MAX	a
1000	0
10000	9212
100000	-54516
1000000	-228326

Los resultados no tienen sentido, desde el 0 hasta los números casi aleatorios obtenidos en los siguientes valores de MAX, esto se debe a la condición de carrera provocada por la falta de sincronización en el acceso a la variable compartida a, esta falta de sincronización se debe a que no hay una forma de controlarlo, que en este caso serían los semaforos

3.2 Código con semáforos

Ejecutar varias ocasiones el código con semáforos con la variable MAX igual a 1000000, 1e9, 1e12.
Completar la tabla 2 con el valor de la variable a que se imprime en pantalla.
Comentar por qué se obtienen los resultados.

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

//Variable compartida
int a=0;
long MAX=1000000000;

void * funcion_hilo1(void *arg);
void * funcion_hilo2(void *arg);

//Declarar el semáforo
sem_t s;

int main (void)
{
        pthread_t hilo1, hilo2;
        //Inicializa el semáforo s, 0 porque no es compartido entre procesos sino ente hilos de un mismo proceso, valor inicial
        sem_init(&s,0,1);
        pthread_create(&hilo1, NULL, *funcion_hilo1, NULL);
        pthread_create(&hilo2, NULL, *funcion_hilo2, NULL);
        //Para esperar que terminen los hilos
        pthread_join(hilo1, NULL);
        pthread_join(hilo2, NULL);
        printf ("El valor de a es %d \n",a);
        return 0;
}

void * funcion_hilo1(void *arg)
{
        for(int i=0; i<MAX; i++)
        {
                //Bloqueo la variable compartida con sem_wait
                sem_wait(&s);
                a +=1;
                //Incremento el valor del semáforo
                sem_post(&s);
        }
}

void * funcion_hilo2(void *arg)
{
        for(int i=0; i<MAX; i++)
        {
                sem_wait(&s);
                a -=1;
                sem_post(&s);
        }
}
```

felipemerino@localhost:~ — vim ej3_2.c

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

// Variable compartida
int a = 0;

// VALORES A PROBAR:
// 1000000
// 1000000000
// 1000000000000
long MAX = 1000000;

// Declarar el semáforo
sem_t s;

void * funcion_hilo1(void *arg);
void * funcion_hilo2(void *arg);

int main (void) {
    pthread_t hilo1, hilo2;

    // Inicializa el semáforo 's'.
    // 0 = compartido entre hilos del mismo proceso.
    // 1 = valor inicial (semáforo en verde/abierto).
    sem_init(&s, 0, 1);

    // Creación de hilos
    pthread_create(&hilo1, NULL, funcion_hilo1, NULL);
    pthread_create(&hilo2, NULL, funcion_hilo2, NULL);

    // Esperar a que terminen los hilos
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);

    // Destruir el semáforo al terminar
    sem_destroy(&s);

    printf("El valor de a es %d \n", a);
    return 0;
```

I… ej3_2.c[+] c 7% ln : 5 %:23

-- INSERTAR --

```
[felipemerino@localhost ~]$ vim ej3_2.c
[felipemerino@localhost ~]$ gcc -pthread ej3_2.c -std=c99 -o ej3_2
[felipemerino@localhost ~]$ ./ej3_2
```

Figura 2. Código con semáforos

Tabla 2

MAX	a
1e6	0
1e9	0
1e12	0

Se garantizó exclusión mutua con los semáforos, pero aumentó por mucho el tiempo de ejecución de los scripts, pero todos los resultados dieron 0, por eso podemos notar que no existieron los problemas de condición de carrera vistos en 3.1

3.3 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.

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

// Variable compartida
int a = 0;


long MAX = 10;

sem_t s;

void * funcion_hilo1(void *arg);
void * funcion_hilo2(void *arg);
void * funcion_hilo3(void *arg); // Declaración del tercer hilo

int main (void) {
    pthread_t hilo1, hilo2, hilo3;

    // Inicializamos semáforo en 1 (verde/libre)
    sem_init(&s, 0, 1);
    pthread_create(&hilo1, NULL, funcion_hilo1, NULL);
    pthread_create(&hilo2, NULL, funcion_hilo2, NULL);

    pthread_create(&hilo3, NULL, funcion_hilo3, NULL);

    // Esperamos a los 3 hilos
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);
    pthread_join(hilo3, NULL);

    sem_destroy(&s);

    printf("--- FIN DEL PROCESO ---\n");
    printf("El valor final de a es %d \n", a);
    return 0;
}

void * funcion_hilo1(void *arg) {
    int valor_sem;
```

```
40      int valor_sem;
41      for(int i=0; i<MAX; i++) {
42          sem_wait(&s); // Bloqueo (Semáforo se pone en 0)
43
44          // --- SECCIÓN CRÍTICA ---
45          a += 1;
46
47          // Obtener valor del semáforo
48          sem_getvalue(&s, &valor_sem);
49          printf("Hilo 1 (Suma)  | Semáforo: %d | a = %d\n", valor_sem, a);
50          // ----------------------
51
52          sem_post(&s); // Desbloqueo (Semáforo vuelve a 1)
53      }
54      return NULL;
55  }
56
57  void * funcion_hilo2(void *arg) {
58      int valor_sem;
59      for(int i=0; i<MAX; i++) {
60          sem_wait(&s);
61
62          // --- SECCIÓN CRÍTICA ---
63          a -= 1;
64
65          sem_getvalue(&s, &valor_sem);
66          printf("Hilo 2 (Resta) | Semáforo: %d | a = %d\n", valor_sem, a);
67
68
69          sem_post(&s);
70      }
71      return NULL;
72  }
73
74  void * funcion_hilo3(void *arg) {
75      int valor_sem;
76      for(int i=0; i<MAX; i++) {
77          sem_wait(&s);
78
79
```

```
74 void * funcion_hilo3(void *arg) {
75     int valor_sem;
76     for(int i=0; i<MAX; i++) {
77         sem_wait(&s);
78 
79 
80         a += 1;
81 
82         sem_getvalue(&s, &valor_sem);
83         printf("Hilo 3 (Suma)  | Semáforo: %d | a = %d\n", valor_sem, a);
84 
85 
86         sem_post(&s);
87     }
88     return NULL;
89 }
```

```
[felipemerino@localhost ~]$ vim ej3_3.c
[felipemerino@localhost ~]$ gcc -pthread ej3_3.c -std=c99 -o ej3_3
./ej3_3
Hilo 1 (Suma)   | Semáforo: 0 | a = 1
Hilo 1 (Suma)   | Semáforo: 0 | a = 2
Hilo 1 (Suma)   | Semáforo: 0 | a = 3
Hilo 1 (Suma)   | Semáforo: 0 | a = 4
Hilo 1 (Suma)   | Semáforo: 0 | a = 5
Hilo 1 (Suma)   | Semáforo: 0 | a = 6
Hilo 1 (Suma)   | Semáforo: 0 | a = 7
Hilo 1 (Suma)   | Semáforo: 0 | a = 8
Hilo 1 (Suma)   | Semáforo: 0 | a = 9
Hilo 1 (Suma)   | Semáforo: 0 | a = 10
Hilo 3 (Suma)   | Semáforo: 0 | a = 11
Hilo 3 (Suma)   | Semáforo: 0 | a = 12
Hilo 3 (Suma)   | Semáforo: 0 | a = 13
Hilo 3 (Suma)   | Semáforo: 0 | a = 14
Hilo 3 (Suma)   | Semáforo: 0 | a = 15
Hilo 3 (Suma)   | Semáforo: 0 | a = 16
Hilo 3 (Suma)   | Semáforo: 0 | a = 17
Hilo 3 (Suma)   | Semáforo: 0 | a = 18
Hilo 3 (Suma)   | Semáforo: 0 | a = 19
Hilo 3 (Suma)   | Semáforo: 0 | a = 20
Hilo 2 (Resta)  | Semáforo: 0 | a = 19
Hilo 2 (Resta)  | Semáforo: 0 | a = 18
Hilo 2 (Resta)  | Semáforo: 0 | a = 17
Hilo 2 (Resta)  | Semáforo: 0 | a = 16
Hilo 2 (Resta)  | Semáforo: 0 | a = 15
Hilo 2 (Resta)  | Semáforo: 0 | a = 14
Hilo 2 (Resta)  | Semáforo: 0 | a = 13
Hilo 2 (Resta)  | Semáforo: 0 | a = 12
Hilo 2 (Resta)  | Semáforo: 0 | a = 11
Hilo 2 (Resta)  | Semáforo: 0 | a = 10
--- FIN DEL PROCESO ---
El valor final de a es 10
```

3.4 Código con semáforo. Variable color de texto.

Tomar como base el código de la Figura 3 propuesto y añadir un semáforo que permita compartir la variable que define el color del texto.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void * rojo(void *id)
{
#define A "\x1b[31m"
printf (A "Este texto es ROJO! \n");
}

void * verde(void *id)
{
#define B "\x1b[32m"
printf(B  "Este texto es VERDE! \n");
}

int main()
{
pthread_t hilo_rojo, hilo_verde;
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");
return 0;
}
```

Figura 3. Script de ejemplo para observar la variable "Color de texto"

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

// Declaramos el semáforo globalmente
sem_t s;

void * rojo(void *id)
{
    // Bloqueamos antes de imprimir para proteger el recurso (consola/color)
    sem_wait(&s);

    #define A "\x1b[31m"
    printf (A "Este texto es ROJO! \n");

    // Liberamos el semáforo
    sem_post(&s);
    return NULL;
}

void * verde(void *id)
{
    sem_wait(&s);

    #define B "\x1b[32m"
    printf(B "Este texto es VERDE! \n");

    sem_post(&s);
    return NULL;
}

int main()
{
    pthread_t hilo_rojo, hilo_verde;

    // Inicializamos el semáforo: 0 (hilos), 1 (valor inicial disponible)
    sem_init(&s, 0, 1);

    pthread_create(&hilo_rojo, NULL, *rojo, NULL);
```

```
INSERT  ej3_4.c[+]
- INSERTAR --
```

```
41      pthread_create(&hilo_verde, NULL, *verde, NULL);
42
43      pthread_join(hilo_rojo, NULL);
44      pthread_join(hilo_verde, NULL);
45
46      #define C "\x1b[34m"
47      printf (C "Este texto es AZUL \n");
48
49      // Destruimos el semáforo al final
50      sem_destroy(&s);
51
52      return 0;
53  }
INSERT  ej3_4.c[+]
```

```
felipemerino@localhost ~]$ vim ej3_4.c
felipemerino@localhost ~]$ gcc -pthread ej3_4.c -std=c99 -o ej3_4
felipemerino@localhost ~]$ ./ej3_4
ste texto es ROJO!
ste texto es VERDE!
ste texto es AZUL
felipemerino@localhost ~]$
```

3.5 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.

```c
#include <pthread.h>
#include <stdio.h>
#define MAX 10000000

int a=0;

void * funcion_hilo1(void *arg)
{

        pthread_mutex_t * mutex=arg;
        pthread_mutex_lock(mutex);
        for(int i=0; i<MAX; i++)
        {
                a+=1;
        }

        pthread_mutex_unlock(mutex);
}
void * funcion_hilo2(void *arg)
{
        pthread_mutex_t * mutex=arg;
        pthread_mutex_lock(mutex);
        for(int i=0; i<MAX; i++)
        {

                a-=1;
        }

        pthread_mutex_unlock(mutex);
}


int main()
{
        pthread_t hilo1, hilo2;
        pthread_mutex_t mutex;
        pthread_mutex_init(&mutex,NULL);
        pthread_create(&hilo1, NULL,funcion_hilo1, &mutex);
        pthread_create(&hilo2, NULL,funcion_hilo2, &mutex);
        pthread_join(hilo1, NULL);
        pthread_join(hilo2, NULL);
        pthread_mutex_destroy(&mutex);
        printf ("El valor de a es %d \n",a);
        return 0;
}
```

Figura 4.Código con mutex

felipemerino@localhost:~ — vim ej3_5.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void * rojo(void *arg)
{
    // Casteamos el argumento de vuelta a un puntero de mutex
    pthread_mutex_t * mutex = (pthread_mutex_t *) arg;

    // Bloqueo del mutex (Sección Crítica)
    pthread_mutex_lock(mutex);

    #define A "\x1b[31m"
    printf (A "Este texto es ROJO! \n");

    // Desbloqueo del mutex
    pthread_mutex_unlock(mutex);
    return NULL;
}

void * verde(void *arg)
{
    // Casteamos el argumento
    pthread_mutex_t * mutex = (pthread_mutex_t *) arg;

    pthread_mutex_lock(mutex);

    #define B "\x1b[32m"
    printf(B "Este texto es VERDE! \n");

    pthread_mutex_unlock(mutex);
    return NULL;
}
```

```c
int main()
{
    pthread_t hilo_rojo, hilo_verde;

    // Declaración e inicialización del Mutex (como en Figura 4)
    pthread_mutex_t mutex;
    pthread_mutex_init(&mutex, NULL);

    // Pasamos la dirección del mutex (&mutex) como último argumento
    pthread_create(&hilo_rojo, NULL, *rojo, &mutex);
    pthread_create(&hilo_verde, NULL, *verde, &mutex);

    pthread_join(hilo_rojo, NULL);
    pthread_join(hilo_verde, NULL);

    // Destruir el mutex al finalizar
    pthread_mutex_destroy(&mutex);

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

    return 0;
}
```

```
felipemerino@localhost ~]$ vim ej3_5.c
felipemerino@localhost ~]$ gcc -pthread ej3_5.c -std=c99 -o ej3_5
felipemerino@localhost ~]$ ./ej3_5
Este texto es ROJO!
Este texto es VERDE!
Este texto es AZUL
felipemerino@localhost ~]$
```

4. CONCLUSIONES Y RECOMENDACIONES

Dentro de la práctica se evidenció cómo la falta de sincronización de procesos causa condiciones de carrera (las inconsistencias que se notaron en la tabla 3.1), con el uso de semáforos y mutex se corrigió el problema garantizando exclusión mutua verificando resultados consistentes en dichas operaciones