


	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 1</p>

## INFORME DE LABORATORIO

### (formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	Sistemas Operativos				
TÍTULO DE LA PRÁCTICA:	<i>Threads en C- PThreads</i>				
NÚMERO DE PRÁCTICA:	06	AÑO LECTIVO:	2023	NRO. SEMESTRE:	05
FECHA DE PRESENTACIÓN	12/06/2023	HORA DE PRESENTACIÓN	—		
<b>INTEGRANTE (s):</b> Yoset Cozco Mauri				<b>NOTA:</b>	
<b>DOCENTE(s):</b> NIETO VALENCIA, RENE ALONSO					

SOLUCIÓN Y RESULTADOS
<p>I. Ejercicio Resuelto</p> <p>1. PThreads.</p> <p>Code01.c</p> <pre>#include &lt;stdio.h&gt; #include &lt;pthread.h&gt;  void* funcion(void* p1); int c = 0;  int main() {     pthread_t hilo;     pthread_attr_t attr;     int error;      pthread_attr_init(&amp;attr);</pre>

```
error = pthread_create(&hilo, &attr, funcion, NULL);
if (error != 0) {
    perror("error");
    return(-1);
}

int i = 0;
while (i < 300) {
    c++;
    printf("Padre: %d\n", c);
    i++;
}

// No call to pthread_join here
return 0;
}

void* funcion(void* p1) {
    int i = 0;
    while (i < 300) {
        c--;
        printf("Hijo: %d\n", c);
        i++;
    }
    pthread_exit(0);
}
```

- **Ejecucion de codigo:**

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Padre: -27
Padre: -26
Padre: -25
Padre: -24
Padre: -23
Padre: -22
Padre: -21
Padre: -20
Padre: -19
Padre: -18
Padre: -17
Padre: -16
Padre: -15
Padre: -14
Padre: -13
Padre: -12
Padre: -11
Padre: -10
Padre: -9
Padre: -8
Padre: -7
Padre: -6
Padre: -5
Padre: -4
Padre: -3
Padre: -2
Padre: -1
Padre: 0
```

```
○ tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ ss
```

- Analice el código del ejemplo la sección PThreads. Describa qué actividad realiza el código que se muestra. Detalle los comandos utilizados para poder ejecutar de forma correcta el código.

El código utiliza la biblioteca PThreads en C para crear dos hilos que ejecutan tareas concurrentes. El hilo principal incrementa una variable global llamada c en un bucle while, mientras que el otro hilo (llamado función) la decrementa en otro bucle while.

Comandos usados:

- gcc -o code01 code01.c -lpthread
- ./code01

- **Explique a qué se debe el orden de ejecución del proceso principal y del Thread creado, ¿Se ejecutan en paralelo o de forma secuencial?**

El orden de ejecución puede ser en paralelo, donde los hilos se ejecutan simultáneamente, o de forma secuencial, donde se alternan en la ejecución. Esto depende del sistema operativo y la planificación de hilos utilizada para administrar los recursos del sistema.

- **Del código de la sección PThreads, qué ocurre si se remueve la sentencia pthread\_exit(0).**

Elabore un ejemplo de código que demuestre el comportamiento del código sin esta sentencia.

La sentencia pthread\_exit(0) es fundamental para una correcta finalización de los hilos. Su ausencia puede provocar comportamientos inesperados y una terminación abrupta del programa sin garantizar la finalización adecuada de los hilos secundarios.

- **De igual manera, involucra remover la sentencia pthread\_join(hilo, NULL). ¿Existen cambios significativos en el comportamiento de los códigos? Elabore un ejemplo en código.**

Si se remueve la sentencia pthread\_join(hilo, NULL) del código, habrá cambios significativos en el comportamiento del programa. La función pthread\_join() se utiliza para esperar a que un hilo específico finalice antes de que el hilo principal continúe su ejecución.

**Ejemplo de código:** #include <stdio.h>

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

```
void* funcion(void* p1);
```

```
int c = 0;
```

```
int main() {
```

```
    pthread_t hilo;
```

```
    pthread_attr_t attr;
```

```
    int error;
```

```
    pthread_attr_init(&attr);
```

```
    error = pthread_create(&hilo, &attr, funcion, NULL);
```

```
    if (error != 0) {
```

```
        perror("error");
```

```
        return -1;
```

```
}

int i = 0;
while (i < 300) {
    c++;
    printf("Padre: %d\n", c);
    i++;
}

// La sentencia pthread_join() se removio

return 0;
}

void* funcion(void* p1) {
    int i = 0;
    while (i < 300) {
        c--;
        printf("Hijo: %d\n", c);
        i++;
    }
    pthread_exit(0);
}

// code01_incompleto.c end
// how to execute with lpthread:
// gcc -o code01_incompleto code01_incompleto.c -lpthread
```

```
38     pthread_exit(0);
39 }
40 // code01_incompleto.c end
41 // how to execute with lpthread:
42 // gcc -o code01_incompleto code01_incompleto.c -lpthread
43
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Padre: -27
Padre: -26
Padre: -25
Padre: -24
Padre: -23
Padre: -22
Padre: -21
Padre: -20
Padre: -19
Padre: -18
Padre: -17
Padre: -16
Padre: -15
Padre: -14
Padre: -13
Padre: -12
Padre: -11
Padre: -10
Padre: -9
Padre: -8
Padre: -7
Padre: -6
Padre: -5
Padre: -4
Padre: -3
Padre: -2
Padre: -1
Padre: 0
```

```
tyy3@tyy3:~/Desktop/2023/nsa23_02/so_lab/lab06$
```

- En el siguiente ejercicio se intenta medir el tiempo de creación de 100 procesos (pesados) a partir de un proceso padre, y el tiempo de creación de 100 hilos. Observe los resultados de ambos programas. Haga pruebas con un valor mayor de procesos y de hilos. ¿Qué conclusiones le merece los resultados observados?

Code02.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

int main() {
    struct timeval t0, t1;
    int i = 0;
    int id = -1;
    gettimeofday(&t0, NULL);
    for (i = 0; i < 100; i++) {
        id = fork();
        if (id == 0) return 0;
    }
    if (id != 0) {
        gettimeofday(&t1, NULL);
        unsigned int ut1 = t1.tv_sec * 1000000 + t1.tv_usec;
        unsigned int ut0 = t0.tv_sec * 1000000 + t0.tv_usec;
        /* Tiempo medio en microsegundos */
        printf("%f\n", (ut1 - ut0) / 100.0);
    }
    return 0;
}

// exec code: gcc code02.c -o code02
```

```

30 |
31 |         gettimeofday(&t1, NULL);
32 |         unsigned int ut1 = t1.tv_sec * 1000000 + t1.tv_usec;
33 |         unsigned int ut0 = t0.tv_sec * 1000000 + t0.tv_usec;
34 |         /* Tiempo medio en microsegundos */
35 |         printf("%f\n", (ut1 - ut0) / 100.0);
36 |     }
37 |
38 |     return 0;
39 | }
40 | // exec code: gcc code02.c -o code02

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc code02.c -o code02
code02.c: In function 'main':
code02.c:11:14: warning: implicit declaration of function 'fork' [-Wimplicit-function-decl
11 |         id = fork();
    |         ^~~~~
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc code02.c -o code02
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc code02.c -o code02
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ ./code02
41.160000
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$

```

Code03.c:

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>

struct timeval t0, t1;
double media = 0.0;

```



```
void *hilo(void *arg) {
    gettimeofday(&t1, NULL);
    unsigned int ut1 = t1.tv_sec * 1000000 + t1.tv_usec;
    unsigned int ut0 = t0.tv_sec * 1000000 + t0.tv_usec;
    media += (ut1 - ut0);
    pthread_exit(NULL);
}

int main() {
    int i = 0;
    pthread_t h;

    for (i = 0; i < 100; i++) {
        gettimeofday(&t0, NULL);
        pthread_create(&h, NULL, hilo, NULL);
        pthread_join(h, NULL);
    }

    /* Tiempo medio en microsegundos */
    printf("%f\n", (media / 100.0));

    return 0;
}

//exec code: gcc code03.c -o code03 -lpthread
```

#### PRUEBAS:

```
41.100000
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc code03.c -o code03 -lpthread
code03.c: In function 'hilo':
code03.c:10:5: warning: implicit declaration of function 'gettimeofday' [-Wimplicit-function-declaration]
   10 |     gettimeofday(&t1, NULL);
      |     ^~~~~~
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ ./code03
101.540000
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$
```

- Analice el código del siguiente programa. Describa qué actividad realiza el código que se muestra. Agregue una función que sea invocada por el hilo principal para mostrar el contenido de la matriz antes y después de su modificación.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

struct parametros {
    int id;
    float escalar;
    float matriz[3][3];
};

void init(float m[3][3]) {
    int i;
    int j;
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            m[i][j] = (float)rand() / RAND_MAX * 100;
        }
    }
}

void* matrizporescalar(void* arg) {
    struct parametros* p;
    int i;
    int j;
    p = (struct parametros*)arg;
    for (i = 0; i < 3; i++) {
        printf("Hilo %d multiplicando fila %d\n", p->id, i);
        for (j = 0; j < 3; j++) {
            p->matriz[i][j] = p->matriz[i][j] * p->escalar;
        }
    }
}
```

```
        sleep(5);
    }
}
return NULL;
}

void mostrarMatriz(float m[3][3]) {
    int i, j;
    printf("Contenido de la matriz:\n");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            printf("%.2f ", m[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

int main(int argc, char* argv[]) {
    pthread_t h1;
    struct parametros p1;
    p1.id = 1;
    p1.escalar = 5.0;
    init(p1.matriz);

    mostrarMatriz(p1.matriz); // Mostrar matriz antes de la modificación

    pthread_create(&h1, NULL, matrizporescalar, (void*)&p1);
    pthread_join(h1, NULL);

    mostrarMatriz(p1.matriz); // Mostrar matriz después de la modificación

    printf("Fin\n");
}
```

```
return 0;
```

```
}
```

```
// exec code: gcc -o matriz matriz.c -pthread
```

pruebas :

```
21  }
22
23  void* matrizporescalar(void* arg) {
24      struct parametros* p;
25      int i;
26      int j;
27      p = (struct parametros*)arg;
28      for (i = 0; i < 3; i++) {
29          printf("Hilo %d multiplicando fila %d\n", p->id, i);
30          for (j = 0; j < 3; j++) {
31              p->matriz[i][j] = p->matriz[i][j] * p->escalar;
32              sleep(5);
33          }
34      }
35      return NULL;
36  }
37
38  void mostrarMatriz(float m[3][3]) {
39      int i, j;
40      printf("Contenido de la matriz:\n");
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

```
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc -o matriz matriz.c -pthread
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ ./matriz
Contenido de la matriz:
84.02 39.44 78.31
79.84 91.16 19.76
33.52 76.82 27.78

Hilo 1 multiplicando fila 0
Hilo 1 multiplicando fila 1
Hilo 1 multiplicando fila 2
Contenido de la matriz:
420.09 197.19 391.55
399.22 455.82 98.78
167.61 384.11 138.89

Fin
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$
```

## II. Ejercicios Propuestos.

1. Modifique el programa Ejercicio\_Matriz de tal forma que, un hilo se encargue de multiplicar cada fila de la matriz por un escalar cualquiera N.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

struct parametros {
    int id;
    float matriz[3][3];
};

void init(float m[3][3]) {
    int i, j;
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            m[i][j] = (float)rand() / RAND_MAX * 100;
        }
    }
}

void* matrizporescalar(void* arg) {
    struct parametros* p;
    int i, j;
    float escalar = *((float*)arg); // Obtiene el escalar N del argumento pasado al
hilo
    p = (struct parametros*)arg;
    for (i = 0; i < 3; i++) {
        printf("Hilo %d multiplicando fila %d\n", p->id, i);
        for (j = 0; j < 3; j++) {
            p->matriz[i][j] = p->matriz[i][j] * escalar; // Multiplica cada elemento
por el escalar N
        }
    }
}
```

```
        sleep(5);
    }
}
return NULL;
}

void mostrarMatriz(float m[3][3]) {
    int i, j;
    printf("Contenido de la matriz:\n");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            printf("%.2f ", m[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

int main(int argc, char* argv[]) {
    pthread_t h1;
    struct parametros p1;
    float escalar = 5.0;
    p1.id = 1;
    init(p1.matriz);

    mostrarMatriz(p1.matriz); // Mostrar matriz original

    pthread_create(&h1, NULL, matrizporescalar, (void*)&escalar); // Pasa el escalar N
    como argumento al hilo
    pthread_join(h1, NULL);

    mostrarMatriz(p1.matriz); // Mostrar matriz modificada

    printf("Fin\n");
}
```



UNIVERSIDAD NACIONAL DE SAN AGUSTIN  
FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS  
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA



**Formato:** Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

**Aprobación:** 2022/03/01

**Código:** GUIA-PRLE-001

**Página:** 15

```
    return 0;  
}  
// exec code: gcc matriz.c -o matriz -lpthread
```

Pruebas:

```

21 }
22
23 void* matrizporescalar(void* arg) {
24     struct parametros* p;
25     int i;
26     int j;
27     p = (struct parametros*)arg;
28     for (i = 0; i < 3; i++) {
29         printf("Hilo %d multiplicando fila %d\n", p->id, i);
30         for (j = 0; j < 3; j++) {
31             p->matriz[i][j] = p->matriz[i][j] * p->escalar;
32             sleep(5);
33         }
34     }
35     return NULL;
36 }
37
38 void mostrarMatriz(float m[3][3]) {
39     int i, j;
40     printf("Contenido de la matriz:\n");

```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

```

• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc -o matriz matriz.c -pthread
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ ./matriz
Contenido de la matriz:
84.02 39.44 78.31
79.84 91.16 19.76
33.52 76.82 27.78

Hilo 1 multiplicando fila 0
Hilo 1 multiplicando fila 1
Hilo 1 multiplicando fila 2
Contenido de la matriz:
420.09 197.19 391.55
399.22 455.82 98.78
167.61 384.11 138.89

Fin
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$

```

- Modifique el programa Ejercicio\_Matriz de modo que a través de otro hilo se muestre el contenido de la matriz.

Codigo:

```

#include <stdio.h>
#include <stdlib.h>

```



```
#include <pthread.h>
#include <time.h>

#define SIZE 3

struct parametros {
    int id;
    float matriz[SIZE][SIZE];
};

void init(float m[SIZE][SIZE]) {
    srand((unsigned int) time(NULL)); // Semilla para generar números aleatorios
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            m[i][j] = (float)rand() / RAND_MAX * 100;
        }
    }
}

void mostrarMatriz(float m[SIZE][SIZE]) {
    printf("Contenido de la matriz:\n");
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            printf("%.2f ", m[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

void* matrizporescalar(void* arg) {
    struct parametros* p = (struct parametros*)arg;
    float escalar = 5.0;
    for (int i = 0; i < SIZE; i++) {
```

```
        printf("Hilo %d multiplicando fila %d\n", p->id, i);
        for (int j = 0; j < SIZE; j++) {
            p->matriz[i][j] *= escalar;
        }
    }
    return NULL;
}

int main(int argc, char* argv[]) {
    pthread_t h1;
    struct parametros p1;
    p1.id = 1;
    init(p1.matriz);

    mostrarMatriz(p1.matriz); // Mostrar matriz original

    pthread_create(&h1, NULL, matrizporescalar, (void*)&p1);
    pthread_join(h1, NULL);

    mostrarMatriz(p1.matriz); // Mostrar matriz modificada

    printf("Fin\n");

    return 0;
}
```

Pruebas:

```

40         for (j = 0; j < 3; j++) {
41             printf("%.2f ", m[i][j]);
42         }
43         printf("\n");
44     }
45     printf("\n");
46 }
47
48 int main(int argc, char* argv[]) {
49     pthread_t h1;
50     struct parametros p1;

```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

```

• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc matriz_m01.c -o matriz_m01 -lpthread
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ ./matriz_m01
Contenido de la matriz:
84.02 39.44 78.31
79.84 91.16 19.76
33.52 76.82 27.78

Hilo 1084227584 multiplicando fila 0
Hilo 1084227584 multiplicando fila 1
Hilo 1084227584 multiplicando fila 2
Contenido de la matriz:
420.09 197.19 391.55
399.22 455.82 98.78
33.52 76.82 27.78

Fin
○ tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ s

```

- Elabore un programa con un mínimo de 4 Threads, que realice la suma de una lista de números (esta estructura contendrá más de 5000 elementos generados de forma aleatoria).

Codigo 4 hilos:

```

4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <pthread.h>
7. #include <time.h>
8.
9. #define NUM_THREADS 4
10. #define NUM_ELEMENTS 5000

```

```
11.
12. // Estructura para pasar parámetros al hilo
13. typedef struct {
14.     int start; // índice de inicio en el array para el hilo
15.     int end;   // índice final en el array para el hilo
16.     int* array; // puntero al array
17.     long long sum; // suma de elementos procesados por el hilo
18. } ThreadData;
19.
20. // Función que será ejecutada por cada hilo
21. void* sum_array(void* arg) {
22.     ThreadData* data = (ThreadData*) arg;
23.     data->sum = 0;
24.     for (int i = data->start; i < data->end; ++i) {
25.         data->sum += data->array[i];
26.     }
27.     return NULL;
28. }
29.
30. int main() {
31.     srand((unsigned int) time(NULL)); // Semilla para números aleatorios
32.
33.     // Crear y llenar el array con números aleatorios
34.     int array[NUM_ELEMENTS];
35.     for (int i = 0; i < NUM_ELEMENTS; ++i) {
36.         array[i] = rand() % 100; // Números aleatorios entre 0 y 99
37.     }
38.
```

```
39. pthread_t threads[NUM_THREADS];
40. ThreadData threadData[NUM_THREADS];
41.
42. // Crear los hilos
43. for (int i = 0; i < NUM_THREADS; ++i) {
44.     threadData[i].start = i * (NUM_ELEMENTS / NUM_THREADS);
45.     threadData[i].end = (i + 1) * (NUM_ELEMENTS / NUM_THREADS);
46.     threadData[i].array = array;
47.     pthread_create(&threads[i], NULL, sum_array, &threadData[i]);
48. }
49.
50. // Esperar a que los hilos terminen
51. for (int i = 0; i < NUM_THREADS; ++i) {
52.     pthread_join(threads[i], NULL);
53. }
54.
55. // Sumar los resultados de cada hilo
56. long long totalSum = 0;
57. for (int i = 0; i < NUM_THREADS; ++i) {
58.     totalSum += threadData[i].sum;
59. }
60.
61. printf("Suma total: %lld\n", totalSum);
62.
63. return 0;
64. }
65.
66. // Exec code:
```

```
67. // gcc -o threaded_sum threaded_sum.c -pthread
68. // ./threaded_sum
69.
```

Prueba:

```
Fin
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc -o threaded_sum threaded_sum.c -pthread
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ ./threaded_sum
Suma total: 248942
○ tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ s
```

Código 1 solo hilo:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NUM_ELEMENTS 5000

int main() {
    srand((unsigned int) time(NULL)); // Semilla para números aleatorios

    // Crear y llenar el array con números aleatorios
    int array[NUM_ELEMENTS];
    for (int i = 0; i < NUM_ELEMENTS; ++i) {
        array[i] = rand() % 100; // Números aleatorios entre 0 y 99
    }

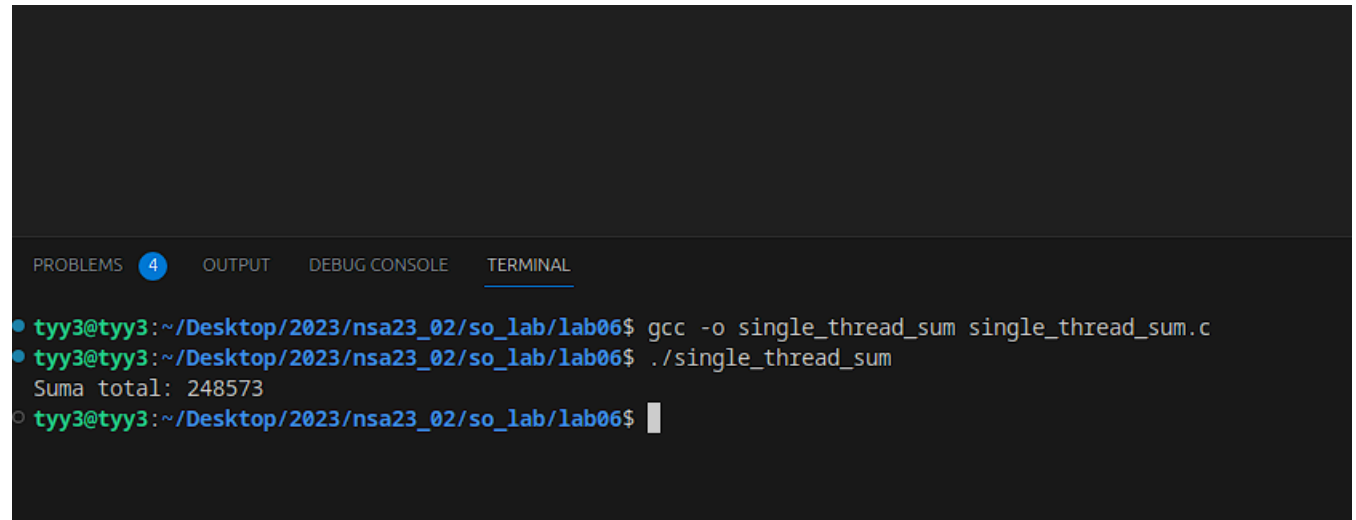
    // Sumar los elementos del array de forma secuencial
    long long totalSum = 0;
    for (int i = 0; i < NUM_ELEMENTS; ++i) {
        totalSum += array[i];
    }

    printf("Suma total: %lld\n", totalSum);
}
```

```
return 0;
}

// Exec code:
// gcc -o single_thread_sum single_thread_sum.c
// ./single_thread_sum
```

### Pruebas:



```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc -o single_thread_sum single_thread_sum.c
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ ./single_thread_sum
Suma total: 248573
○ tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$
```

Elabore un programa que realice el proceso de forma secuencial para validar los resultados obtenidos.

Prueba:

Mida los tiempos requeridos por ambos programas (con varios hilos/un solo hilo).

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

#define NUM_ELEMENTS 5000

// Variable para decidir cuántos hilos se utilizarán
```

```
// 1 para un solo hilo, >1 para múltiples hilos
int num_threads = 1;

typedef struct {
    int start;
    int end;
    int* array;
    long long sum;
} ThreadData;

void* sum_array(void* arg) {
    ThreadData* data = (ThreadData*) arg;
    data->sum = 0;
    for (int i = data->start; i < data->end; ++i) {
        data->sum += data->array[i];
    }
    return NULL;
}

int main() {
    srand((unsigned int) time(NULL));

    int array[NUM_ELEMENTS];
    for (int i = 0; i < NUM_ELEMENTS; ++i) {
        array[i] = rand() % 100;
    }

    long long totalSum = 0;
    pthread_t threads[num_threads];
    ThreadData threadData[num_threads];

    // Crear los hilos
    for (int i = 0; i < num_threads; ++i) {
        threadData[i].start = i * (NUM_ELEMENTS / num_threads);
```



```
threadData[i].end = (i + 1) * (NUM_ELEMENTS / num_threads);
threadData[i].array = array;
pthread_create(&threads[i], NULL, sum_array, &threadData[i]);
}

// Esperar a que los hilos terminen
for (int i = 0; i < num_threads; ++i) {
    pthread_join(threads[i], NULL);
    totalSum += threadData[i].sum;
}

printf("Suma total: %lld\n", totalSum);

return 0;
}

// Exec code:
// gcc -o sequential_sum sequential_sum.c -pthread
// ./sequential_sum
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

```
...
... • tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ gcc -o sequential_sum sequential_sum.c -pthread
... • tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$ ./sequential_sum
Suma total: 249757
○ tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab06$
```

¿A qué conclusiones puede usted llegar después de esta experiencia?

El uso de múltiples hilos puede ser beneficioso para tareas que se pueden dividir en subprocesos independientes, ya que permite un procesamiento más rápido al aprovechar los núcleos múltiples de la

CPU. Sin embargo, para tareas secuenciales donde cada paso depende del anterior, el uso de múltiples hilos no ofrece beneficios y puede complicar el código.

Al generar semillas para números aleatorios, es importante en el contexto de pruebas y replicabilidad.

### **Cuestionario:**

#### **1. ¿Qué diferencia existe entre crear un proceso con PThread y Fork?**

PThread se utiliza para la concurrencia dentro de un proceso existente, mientras que Fork se utiliza para crear procesos independientes que pueden ejecutar diferentes tareas en paralelo.

#### **2. ¿Por qué no es posible compartir una variable declarada dentro de la función main( ) entre varios PThreads?**

No es posible compartir una variable declarada dentro de la función main() entre varios hilos (PThread) debido a que cada hilo tiene su propio espacio de pila y contexto de ejecución. Esto significa que cada hilo tiene su propia copia de las variables locales, incluida la variable declarada en main().

#### **3. Si existen 2 PThreads que modifican una variable global (uno realiza la operación $x += 1$ y el otro $x -= 1$ , repitiendo 100 veces cada operación ) y se muestra cada operación en pantalla. ¿Se llega a obtener siempre el mismo resultado o este cambia en cada ejecución? Escriba un programa y realice varias pruebas que sustenten sus afirmaciones.**

El resultado puede variar en cada ejecución dado que el comportamiento de la concurrencia depende del orden de ejecución de los hilos y puede haber condiciones de carrera, donde los hilos compiten por acceder y modificar la misma variable al mismo tiempo.

### **I. CONCLUSIONES**

En este laboratorio de sistemas operativos, hemos dado un vistazo a cómo usar hilos para hacer más eficiente el procesamiento de datos. Primero, vimos un ejemplo donde múltiples hilos se ponen a trabajar en paralelo para sumar los elementos de una lista grande. Luego, exploramos cómo hacer el mismo trabajo, pero en un solo hilo para comparar la diferencia. En conclusión usar múltiples hilos puede ser súper útil, pero no es siempre la solución mágica. Es como tener un equipo de trabajo: puede ser genial si lo necesitas, pero a veces con una sola persona dedicada es suficiente. Y sobre los números aleatorios, es bueno tener el control de cuán “al azar” queremos que sean.



UNIVERSIDAD NACIONAL DE SAN AGUSTIN  
FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS  
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA



**Formato:** Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

**Aprobación:** 2022/03/01

**Código:** GUIA-PRLE-001

**Página:** 27

## REFERENCIAS Y BIBLIOGRAFÍA

1. *Programming in C* de Stephen G. Kochan, 4ta edición, Addison-Wesley Professional, 2014
2. *Modern Operating Systems* de Andrew S. Tanenbaum y Herbert Bos, 4ta edición, Pearson, 2014
3. *POSIX Threads Programming* de Blaise Barney, Lawrence Livermore National Laboratory, 2010