


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

SOLUCIÓN Y RESULTADOS
<p>I. Ejercicio Resuelto</p> <p>1. Analice y describa la actividad que realiza el siguiente código. Ponga atención como se muestran los mensajes en el terminal. Explique a que se debe este comportamiento.</p> <pre style="background-color: #2e3436; color: #eeeeec; padding: 10px;">#include &lt;stdio.h&gt; #include &lt;pthread.h&gt;  void* funcion(void* p1); void printResult(char* threadName, int value);  int c = 0;  int main() {     pthread_t hilo;     pthread_attr_t attr;     int error;</pre>

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

if (error != 0) {
    perror("error");
    return (-1);
}

printResult((char*)"proceso Padre", 1);

pthread_join(hilo, NULL);

return 0;
}

void* funcion(void* p11) {
    printResult((char*)"Proceso Hijo:", -1);
    pthread_exit(0);
}

void printResult(char* threadName, int v) {
    int i = 0;

    while (i < 300) {
        c += v;
        printf("Inicio ");

        for (int j = 0; j < 10000; j++) {} //delay


        printf("%s: %d ", threadName, c);

        for (int j = 0; j < 10000; j++) {} //delay
    }
}
```

```
printf("Finalizacion \n");

    i++;
}
}
```

### Ejecucion:



```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL bash - lab07
Inicio Proceso Hijo:: 9 Finalizacion
Inicio Proceso Hijo:: 8 Finalizacion
Inicio Proceso Hijo:: 7 Finalizacion
Inicio Proceso Hijo:: 6 Finalizacion
Inicio Proceso Hijo:: 5 Finalizacion
Inicio Proceso Hijo:: 4 Finalizacion
Inicio Proceso Hijo:: 3 Finalizacion
Inicio Proceso Hijo:: 2 Finalizacion
Inicio Proceso Hijo:: 1 Finalizacion
Inicio Proceso Hijo:: 0 Finalizacion
tyy3@tyy3:~/Desktop/2023/nsa23_02/so_lab/lab07$
```

Explicación: El planificador del sistema operativo es responsable de tomar decisiones sobre qué hilo debe ejecutarse en cada momento y durante cuánto tiempo. Utiliza algoritmos de planificación para determinar el orden de ejecución de los hilos basándose en políticas y prioridades.

2. El siguiente código es similar al anterior, pero contiene una variable de exclusión mutua. Analice y describa la actividad que realiza el siguiente código. Ponga atención como se muestran los mensajes en el terminal. Explique a que se debe este nuevo comportamiento.

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

pthread_mutex_t mutex;

void* funcion(void* p1);
void printResult(char* threadName, int value);

int c = 0;

int main() {
```

```
pthread_t hilo;
pthread_attr_t attr;
int error;

pthread_mutex_init(&mutex, NULL);
pthread_attr_init(&attr);

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

printResult((char*)"proceso Padre", 1);

pthread_join(hilo, NULL);

return (0);
}

void* funcion(void* p11) {
    printResult((char*)"Proceso Hijo:", -1);
    pthread_exit(0);
}

void printResult(char* threadName, int v) {
    int i = 0;
    while (i < 300) {
        c += v;

        pthread_mutex_lock(&mutex);
        printf("Inicio ");
        for (int j = 0; j < 10000; j++) {} //delay
        printf("%s: %d ", threadName, c);
```

```
        for (int j = 0; j < 10000; j++) {} //delay
        printf("Finalizacion \n");
        pthread_mutex_unlock(&mutex);

        i++;
    }
}
```

#### Analisis y descripcion:

Existe ausencia de exclusión mutua, en este código, y se eliminaron las funciones relacionadas, no se garantiza la exclusión mutua en el acceso de la variable c, varios hilos podrían acceder y modificar la variable c simultáneamente.

En este nuevo código no se utiliza la biblioteca pthread.h para la creación y gestión de hilos.

La función función se ha simplificado y ahora solo imprime un mensaje, no hay manipulación de la variable c.

## II. Ejercicios Propuestos.

1. Del programa de suma de números utilizando 4 o más Threads (Laboratorio 6), modifique su estructura para permitir el uso de mutex en su ejecución (puede emplearlo para ir acumulando la suma de cada proceso sobre una variable final de suma)

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

#define NUM_THREADS 4
#define NUM_ELEMENTS 5000

pthread_mutex_t mutex;
long long totalSum = 0;

// Estructura para pasar parámetros al hilo
typedef struct {
    int start; // índice de inicio en el array para el hilo
    int end;   // índice final en el array para el hilo
}
```

```
int* array; // puntero al array
} ThreadData;

// Función que será ejecutada por cada hilo
void* sum_array(void* arg) {
    ThreadData* data = (ThreadData*) arg;
    long long sum = 0;
    for (int i = data->start; i < data->end; ++i) {
        sum += data->array[i];
    }

    pthread_mutex_lock(&mutex); // Adquirir el mutex antes de actualizar totalSum
    totalSum += sum;
    pthread_mutex_unlock(&mutex); // Liberar el mutex después de actualizar totalSum

    return NULL;
}

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
    }

    pthread_t threads[NUM_THREADS];
    ThreadData threadData[NUM_THREADS];

    pthread_mutex_init(&mutex, NULL); // Inicializar el mutex

    // 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);
}

pthread_mutex_destroy(&mutex); // Destruir el mutex

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

return 0;
}

// file name : ejer01.c
// how to exec : gcc -pthread ejer01.c -o ejer01
```

Ejecucion:

```
51     threadData[i].end = (i + 1) * (NUM_ELEMENTS / NUM_THREADS);
52     threadData[i].array = array;
53     pthread_create(&threads[i], NULL, sum_array, &threadData[i]);
54 }
55
56 // Esperar a que los hilos terminen
57 for (int i = 0; i < NUM_THREADS; ++i) {
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL

bash - lab07 + ▾ □ ☒ ... ^ ×

```
Inicio Proceso Hijo:: 6 Finalizacion
Inicio Proceso Hijo:: 5 Finalizacion
Inicio Proceso Hijo:: 4 Finalizacion
Inicio Proceso Hijo:: 3 Finalizacion
Inicio Proceso Hijo:: 2 Finalizacion
Inicio Proceso Hijo:: 1 Finalizacion
Inicio Proceso Hijo:: 0 Finalizacion
```

```
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab07$ gcc -pthread ejer01.c -o ejer01
```

```
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab07$ ./ejer01
```

```
Suma total: 249333
```

```
• tty3@tty3:~/Desktop/2023/nsa23_02/so_lab/lab07$
```

Ln 68, Col 49 (31 selected) Spaces: 4 UTF-8 LF {} C ☒ Win32 🔔

2. Elaborar un programa del productor-consumidor, donde una variable servirá como recurso compartido, donde el productor podrá escribir un nuevo dato solo si este se encuentra vacío, caso contrario deberá de esperar. En el caso del consumidos, solo podrá obtener un valor si la variable contiene información.

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

#define BUFFER_SIZE 10

int buffer[BUFFER_SIZE];
int count = 0;

pthread_mutex_t mutex;
pthread_cond_t empty;
pthread_cond_t full;

void* productor(void* arg) {
    int dato = 1;
    while (1) {
```



```
pthread_mutex_lock(&mutex);

// Esperar si el buffer está lleno
while (count == BUFFER_SIZE) {
    pthread_cond_wait(&empty, &mutex);
}

// Escribir el dato en el buffer
buffer[count] = dato;
count++;

printf("Productor produce dato: %d\n", dato);

// Notificar al consumidor que hay un nuevo dato disponible
pthread_cond_signal(&full);

pthread_mutex_unlock(&mutex);

dato++;
}
return NULL;
}

void* consumidor(void* arg) {
    while (1) {
        pthread_mutex_lock(&mutex);

        // Esperar si el buffer está vacío
        while (count == 0) {
            pthread_cond_wait(&full, &mutex);
        }

        // Leer el dato del buffer
        int dato = buffer[count - 1];
```

```
count--;

printf("Consumidor consume dato: %d\n", dato);

// Notificar al productor que hay un espacio vacío en el buffer
pthread_cond_signal(&empty);

pthread_mutex_unlock(&mutex);
}
return NULL;
}

int main() {
    pthread_t hiloProductor, hiloConsumidor;

    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&empty, NULL);
    pthread_cond_init(&full, NULL);

    pthread_create(&hiloProductor, NULL, productor, NULL);
    pthread_create(&hiloConsumidor, NULL, consumidor, NULL);

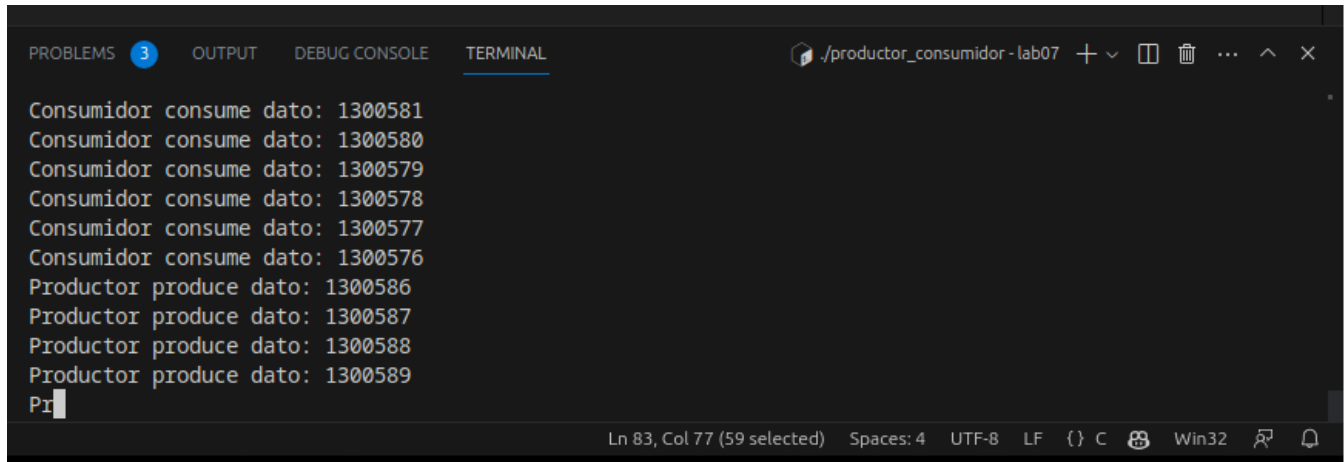
    pthread_join(hiloProductor, NULL);
    pthread_join(hiloConsumidor, NULL);

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&empty);
    pthread_cond_destroy(&full);

    return 0;
}

// file name : productor_consumidor.c
// how to exec : gcc -pthread productor_consumidor.c -o productor_consumidor
```

## Ejecucion:



```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
./productor_consumidor -lab07

Consumidor consume dato: 1300581
Consumidor consume dato: 1300580
Consumidor consume dato: 1300579
Consumidor consume dato: 1300578
Consumidor consume dato: 1300577
Consumidor consume dato: 1300576
Productor produce dato: 1300586
Productor produce dato: 1300587
Productor produce dato: 1300588
Productor produce dato: 1300589
Pr
```

3. Elabore un programa de multiplicación de matrices, que utilice más de 10 Threads para el proceso de multiplicación. (utilice Mutex)

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

#define N 1000
#define M 1000
#define P 1000
#define NUM_THREADS 20

int A[N][M];
int B[M][P];
int C[N][P];

pthread_mutex_t mutex;

typedef struct {
    int startRow;
    int endRow;
} ThreadData;
```

```
void* multiply(void* arg) {
    ThreadData* data = (ThreadData*) arg;

    // Multiplicación de las matrices en el rango de filas asignado al hilo
    for (int i = data->startRow; i < data->endRow; i++) {
        for (int j = 0; j < P; j++) {
            int sum = 0;
            for (int k = 0; k < M; k++) {
                sum += A[i][k] * B[k][j];
            }

            // Adquirir el mutex antes de actualizar la matriz resultante C
            pthread_mutex_lock(&mutex);
            C[i][j] = sum;
            pthread_mutex_unlock(&mutex);
        }
    }

    return NULL;
}

int main() {
    pthread_t threads[NUM_THREADS];
    ThreadData threadData[NUM_THREADS];

    // Inicializar las matrices A y B con valores aleatorios
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            A[i][j] = rand() % 10;
        }
    }

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < P; j++) {
```

```
        B[i][j] = rand() % 10;
    }
}

pthread_mutex_init(&mutex, NULL); // Inicializar el mutex

// Crear hilos para realizar la multiplicación de matrices
int rowsPerThread = N / NUM_THREADS;
for (int i = 0; i < NUM_THREADS; i++) {
    threadData[i].startRow = i * rowsPerThread;
    threadData[i].endRow = (i + 1) * rowsPerThread;

    pthread_create(&threads[i], NULL, multiply, &threadData[i]);
}

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

pthread_mutex_destroy(&mutex); // Destruir el mutex

// Imprimir la matriz resultante C
printf("Matriz resultante C:\n");
for (int i = 0; i < N; i++) {
    for (int j = 0; j < P; j++) {
        printf("%d ", C[i][j]);
    }
    printf("\n");
}

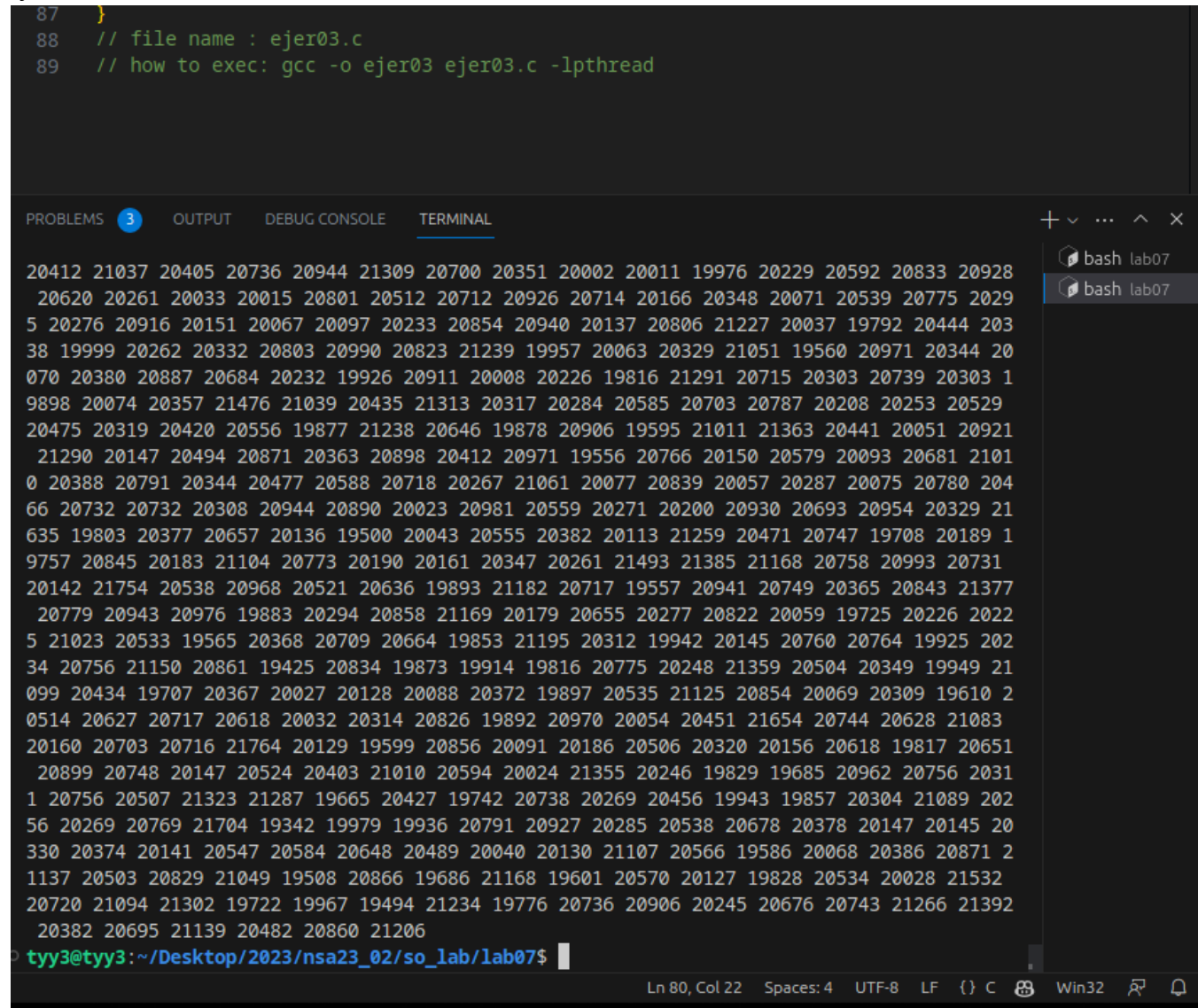
return 0;
}

// file name : ejer03.c
```

```
// how to exec: gcc -o ejer03 ejer03.c -lpthread
```

Ejecución:

```
87 }
88 // file name : ejer03.c
89 // how to exec: gcc -o ejer03 ejer03.c -lpthread
```





### Cuestionario:

1. ¿Cuáles son las ventajas o desventajas encontradas del uso de MUTEX?

**Ventajas:**

**Sincronización de procesos:** mutex nos permite controlar el acceso concurrente a secciones críticas de nuestro código.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 15</p>

**Protección de datos compartidos:** Mutex garantiza que solo un hilo puede modificar o leer los datos en un momento dado.

**Prevención de deadlock:** mutex ayuda a evitar situaciones de deadlock, donde los hilos quedan bloqueados indefinidamente.

**Desventajas:**

**Sobrecarga de rendimiento:** Al adquirir y liberar el mutex los hilos implican cierto costo en términos de tiempo de ejecución.

**Posibilidad de deadlock y condición de carrera:** Aunque los previenen si están mal implementados podrían causar situación de deadlock, si no se liberan adecuadamente después de adquirirlo, además que si no se utiliza en secciones críticas podría generar condiciones de carrera.

**Complejidad de programación:** La utilización de mutex podría ser complicada y propensa a errores.

## I. CONCLUSIONES

En este laboratorio de sistemas operativos, he entendido que el aprendizaje y la aplicación de conceptos como la programación de hilos y el uso de mutex son fundamentales para abordar problemas de concurrencia y paralelismo en el desarrollo de software. Estos conocimientos permiten aprovechar al máximo los recursos de hardware y mejorar el rendimiento de los programas, pero también requieren atención cuidadosa y una comprensión sólida para evitar problemas y garantizar una ejecución confiable y correcta.

## RETROALIMENTACIÓN GENERAL

## • REFERENCIAS Y BIBLIOGRAFÍA

- *Abraham Silberschatz, Greg Gagne, Peter B. Galvin, Operating System Concepts, 10th Edition, 2018 John Wiley & Sons.*
- *William Stallings, Sistema Operativos, Aspectos internos y principios de diseño, 5ta Edición, 2015, Person Prentice Hall*



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:** 16