



Pontificia Universidad
JAVERIANA
Bogotá

Sistemas operativos

T12 - Procesos e Hilos

Grupo 7

Santos Alejandro Arellano Olarte // **Santos-arellano**
Jeison Camilo Alfonso Moreno
Jose Villaroel

12/ Septiembre /2025

T12 - Procesos e Hilos	1
1. INTRODUCCIÓN	4
1.1 Objetivos del Taller	4
1.2 Marco Teórico	4
Procesos en Linux	4
Hilos (Threads)	4
1.3 Metodología	5
2. PARTE 1: DESARROLLO CON PROCESOS	5
2.1 Programa matrizsum.c	5
2.1.1 Descripción Funcional	5
2.1.2 Algoritmo y Estructura	5
2.1.3 Implementación de Procesos	5
2.1.4 Validaciones Implementadas	7
2.1.5 Resultados de Ejecución	7
2.2 Programa ejecutor.c	9
2.2.1 Descripción Funcional	9
2.2.2 Implementación de exec	9
2.2.3 Resultados de Ejecución	10
3. PARTE 2: DESARROLLO CON HILOS	11
3.1 Programa matrizproc.c	11
3.1.1 Descripción Funcional	11
3.1.2 Características Principales	11
3.1.3 Estructura de Datos para Hilos	12
3.1.4 Creación y Gestión de Hilos	12
3.1.5 Distribución Equitativa del Trabajo	13
3.1.6 Resultados de Ejecución con 9 Hilos (Grupo 7)	14
4. ANÁLISIS COMPARATIVO	16
4.1 Procesos vs Hilos	16
4.2 Análisis Específico para el Grupo 7 (9 Hilos)	17
4.3 Ventajas y Desventajas Observadas	18
Procesos (matrizsum)	18
Hilos (matrizproc con 9 hilos)	18
5. VALIDACIONES Y MANEJO DE ERRORES	18
5.1 Validaciones Implementadas	19
5.2 Manejo de Errores	20
6. PRUEBAS REALIZADAS	21
6.1 Plan de Pruebas	21
6.2 Resultados de Pruebas Detalladas	21

Pontificia Universidad Javeriana de Bogotá

6.3 Medición de Rendimiento.....	23
7. DIFICULTADES ENCONTRADAS Y SOLUCIONES.....	24
7.1 Dificultades Técnicas.....	24
7.2 Lecciones Aprendidas.....	25
8. CONCLUSIONES.....	25
8.1 Objetivos Alcanzados.....	25
8.2 Aprendizajes Clave del Grupo 7.....	25
8.3 Aplicaciones Prácticas.....	26
8.4 Reflexión Final del Grupo 7.....	26
9. REFERENCIAS.....	26
ANEXO A: CÓDIGO FUENTE COMPLETO.....	27
A.1 Archivo: Parte 1: matrizsum.c.....	27
A.2 Archivo: Parte 1: ejecutor.c.....	33
A.3 Archivo: Parte 2: matrizproc.c.....	36
ANEXO B: CAPTURAS DE PANTALLA.....	44
B.1 Compilación de los Programas.....	44
B.2 Ejecuciones Exitosas.....	46
B.3 Casos de Error.....	51
B.4 Mediciones de Rendimiento.....	52
ANEXO C: INSTRUCCIONES DE COMPILACIÓN Y EJECUCIÓN.....	53
C.1 Requisitos Previos.....	53
C.2 Compilación.....	53
C.3 Ejecución.....	54
Programa matrizsum.....	54
Programa ejecutor.....	55
Programa matrizproc (9 hilos - Grupo 7).....	55
C.4 Scripts de Prueba.....	56
ANEXO D: ANÁLISIS DE COMPLEJIDAD.....	56
D.1 Complejidad Temporal.....	56
D.2 Complejidad Espacial.....	57

1. INTRODUCCIÓN

1.1 Objetivos del Taller

El presente taller tiene como objetivo principal comprender e implementar el uso básico de procesos e hilos en sistemas operativos Linux. a través del desarrollo de tres programas en lenguaje C, se busca:

- Entender la creación y gestión de procesos mediante la llamada al sistema `fork()`
- Implementar la comunicación y sincronización entre procesos padre e hijos
- Utilizar las llamadas al sistema de la familia `exec` para la ejecución de programas
- Trabajar con hilos POSIX utilizando la librería `pthread`
- Comparar las características, ventajas y desventajas entre procesos e hilos
- aplicar estos conceptos en el procesamiento paralelo de matrices

1.2 Marco Teórico

Procesos en Linux

Un proceso es una instancia de un programa en ejecución. En Linux, cada proceso tiene su propio espacio de direcciones virtuales, lo que proporciona aislamiento y seguridad. La creación de procesos se realiza mediante la llamada al sistema `fork()`, que crea una copia exacta del proceso padre. Esta copia (proceso hijo) recibe un PID (Process ID) diferente y puede ejecutar código independiente.

Los procesos se caracterizan por:

- Espacio de memoria independiente
- Contexto de ejecución propio
- Mayor overhead en creación y cambio de contexto

- Comunicación mediante IPC (Inter-Process Communication)

Hilos (Threads)

Los hilos son unidades de ejecución mas ligeras que comparten el mismo espacio de direcciones del proceso que los contiene. En Linux, se implementan mediante la librería POSIX threads (pthread). Los hilos dentro del mismo proceso pueden acceder a las mismas variables y estructuras de datos, lo que facilita la comunicación pero requiere sincronización para evitar condiciones de carrera.

Los hilos se caracterizan por:

- Espacio de memoria compartido
- Menor overhead en creación y cambio de contexto
- Comunicación directa mediante variables compartidas
- Necesidad de mecanismos de sincronización

1.3 Metodología

El desarrollo del taller se realizó siguiendo una metodología incremental:

1. **análisis de requerimientos:** Estudio detallado de las especificaciones del taller
2. **Diseño de la solución:** Planificación de la arquitectura de cada programa
3. **Implementación:** Codificación en lenguaje C siguiendo buenas practicas
4. **Validación:** Implementación de verificaciones de entrada y manejo de errores
5. **Pruebas:** Ejecución con diferentes casos de prueba
6. **Documentación:** Elaboración del presente informe y documentación del código

2. PaRTE 1: DESaRROLLO CON PROCESOS

2.1 Programa matrizsum.c

2.1.1 Descripción Funcional

El programa `matrizsum.c` implementa el procesamiento paralelo de una matriz cuadrada utilizando múltiples procesos. El programa recibe como argumentos el tamaño de la matriz (m) y el número de procesos a crear (n), validando que n sea divisor de m para una distribución equitativa del trabajo.

2.1.2 algoritmo y Estructura

El algoritmo implementado sigue los siguientes pasos:

1. **Validación de argumentos:** Verifica que se reciban exactamente 2 argumentos y que sean números positivos
2. **Validación de divisibilidad:** Comprueba que n divide exactamente a m
3. **Creación de la matriz:** asigna memoria dinamica y llena la matriz con valores
4. **Creación de procesos:** Utiliza `fork()` para crear n procesos hijos
5. **Distribución del trabajo:** Cada proceso hijo recibe m/n filas para procesar
6. **Procesamiento:** Cada hijo suma los elementos de sus filas asignadas
7. **Sincronización:** El proceso padre espera a todos los hijos con `wait()`

2.1.3 Implementación de Procesos

C/C++

// Fragmento clave de creación de procesos

```
for (int i = 0; i < n; i++) {  
  
    pid_t pid = fork();  
  
    if (pid == 0) { // Proceso hijo  
  
        int inicio_fila = i * filas_por_proceso;  
  
        int fin_fila = inicio_fila + filas_por_proceso;  
  
        printf("Proceso hijo PID=%d procesando filas %d a %d\n",  
               getpid(), inicio_fila, fin_fila - 1);  
  
        // Procesamiento de filas asignadas  
  
        for (int fila = inicio_fila; fila < fin_fila; fila++) {  
  
            int suma = 0;  
  
            printf(" Fila %d: ", fila);  
  
            for (int col = 0; col < m; col++) {  
  
                printf("%d ", matriz[fila][col]);
```

```
        suma += matriz[fila][col];
    }

    printf("-> Suma = %d (PID: %d)\n", suma, getpid());
}

exit(0);
}
}
```

La llamada a `fork()` crea un proceso hijo idéntico al padre. El valor de retorno permite distinguir entre padre (`pid > 0`) e hijo (`pid == 0`). Cada hijo procesa su segmento de la matriz y termina con `exit(0)`.

2.1.4 Validaciones Implementadas

- **Validación de argumentos:** Se verifica cantidad y formato correcto
- **Validación numérica:** Los argumentos deben ser números positivos
- **Validación de divisibilidad:** n debe ser divisor exacto de m
- **Validación de memoria:** Se verifica la asignación exitosa de memoria dinámica
- **Validación de procesos:** Se maneja el error en la creación de procesos

2.1.5 Resultados de Ejecución

Caso 1: Ejecución exitosa con matriz 8x8 y 4 procesos

None

```
$ ./matrizsum 8 4
```

Matriz original (8x8):

```
1 2 3 4 5 6 7 8
```

```
2 3 4 5 6 7 8 9
```

```
3 4 5 6 7 8 9 10
```

4 5 6 7 8 9 10 11

5 6 7 8 9 10 11 12

6 7 8 9 10 11 12 13

7 8 9 10 11 12 13 14

8 9 10 11 12 13 14 15

Proceso hijo PID=15234 procesando filas 0 a 1

Fila 0: 1 2 3 4 5 6 7 8 -> Suma = 36 (PID: 15234)

Fila 1: 2 3 4 5 6 7 8 9 -> Suma = 44 (PID: 15234)

Proceso hijo PID=15235 procesando filas 2 a 3

Fila 2: 3 4 5 6 7 8 9 10 -> Suma = 52 (PID: 15235)

Fila 3: 4 5 6 7 8 9 10 11 -> Suma = 60 (PID: 15235)

Proceso hijo PID=15236 procesando filas 4 a 5

Fila 4: 5 6 7 8 9 10 11 12 -> Suma = 68 (PID: 15236)

Fila 5: 6 7 8 9 10 11 12 13 -> Suma = 76 (PID: 15236)

Proceso hijo PID=15237 procesando filas 6 a 7

Fila 6: 7 8 9 10 11 12 13 14 -> Suma = 84 (PID: 15237)

Fila 7: 8 9 10 11 12 13 14 15 -> Suma = 92 (PID: 15237)

Proceso padre (PID=15233): Todos los hijos han terminado

Caso 2: Error de validación - n no divide a m

None

```
$ ./matrizsum 10 3
```

Error: n (3) debe ser divisor de m (10)

Uso correcto: El número de procesos debe dividir exactamente el tamaño de la matriz

2.2 Programa ejecutor.c

2.2.1 Descripción Funcional

El programa `ejecutor.c` demuestra el uso de las llamadas al sistema de la familia `exec`. Su función es ejecutar el programa `matrizsum` con los argumentos proporcionados, utilizando un proceso hijo.

2.2.2 Implementación de `exec`

C/C++

```
// Uso de execl para ejecutar matrizsum
```

```
pid_t pid = fork();
```

```
if (pid == 0) {
```

```
    // Proceso hijo: ejecutar matrizsum usando execl
```

```
    execl("./matrizsum", "matrizsum", argv[1], argv[2], NULL);
```

```
    // Si llegamos aquí, hubo un error
```

```
    perror("Error ejecutando matrizsum");
```

```
    exit(1);
```

```
}
```

La familia de funciones `exec` reemplaza la imagen del proceso actual con un nuevo programa. En este caso:

- `execl`: Usa una lista de argumentos
- Primer argumento: ruta del ejecutable
- Segundo argumento: nombre del programa (`argv[0]`)
- Argumentos siguientes: parámetros del programa
- `NULL`: marca el fin de la lista de argumentos

2.2.3 Resultados de Ejecución

None

```
$ ./ejecutor 6 3
```

Ejecutando `matrizsum` con argumentos: `m=6, n=3`

Matriz original (6x6):

```
1 2 3 4 5 6
2 3 4 5 6 7
3 4 5 6 7 8
4 5 6 7 8 9
5 6 7 8 9 10
6 7 8 9 10 11
```

Proceso hijo PID=15240 procesando filas 0 a 1

Fila 0: 1 2 3 4 5 6 -> Suma = 21 (PID: 15240)

Fila 1: 2 3 4 5 6 7 -> Suma = 27 (PID: 15240)

Proceso hijo PID=15241 procesando filas 2 a 3

Fila 2: 3 4 5 6 7 8 -> Suma = 33 (PID: 15241)

Fila 3: 4 5 6 7 8 9 -> Suma = 39 (PID: 15241)

Proceso hijo PID=15242 procesando filas 4 a 5

Fila 4: 5 6 7 8 9 10 -> Suma = 45 (PID: 15242)

Fila 5: 6 7 8 9 10 11 -> Suma = 51 (PID: 15242)

Proceso padre (PID=15239): Todos los hijos han terminado

El programa matrizsum terminó con código de salida: 0

3. PARTE 2: DESARROLLO CON HILOS

3.1 Programa matrizproc.c

3.1.1 Descripción Funcional

El programa `matrizproc.c` implementa el procesamiento paralelo de matrices utilizando hilos POSIX. A diferencia del programa con procesos, los hilos comparten el espacio de memoria, permitiendo un acceso directo a la matriz sin necesidad de mecanismos de IPC.

3.1.2 Características Principales

- **Número de hilos:** $p = 9$ (configurado para el Grupo 7: $7 + 2 = 9$)
- **Operaciones soportadas:**
 - "sumar": calcula la suma de elementos de cada fila
 - "max": encuentra el valor máximo de cada fila
- **Distribución de trabajo:** Reparto equitativo de filas entre hilos

3.1.3 Estructura de Datos para Hilos

C/C++

```
typedef struct {
```

```
int **matriz;    // Puntero a la matriz compartida

int m;          // Tamaño de la matriz

int inicio_fila; // Primera fila a procesar

int fin_fila;   // Última fila (exclusiva)

char operacion[10]; // Tipo de operación

int thread_id;  // Identificador del hilo

} ThreadData;
```

Esta estructura permite pasar múltiples parámetros a cada hilo de manera organizada.

3.1.4 Creación y Gestión de Hilos

```
C/C++

// Configuración para Grupo 7

int p = 9; // Grupo 7:  $p = 7 + 2 = 9$ 

// Creación de hilos

pthread_t threads[p];

ThreadData thread_data[p];

for (int i = 0; i < p; i++) {

    // Configurar datos del hilo

    thread_data[i].matriz = matriz;

    thread_data[i].m = m;

    thread_data[i].inicio_fila = fila_actual;
```

```
thread_data[i].fin_fila = fila_actual + filas_este_hilo;

// Crear el hilo

pthread_create(&threads[i], NULL, procesar_filas, &thread_data[i]);
}

// Esperar finalización
for (int i = 0; i < p; i++) {
    pthread_join(threads[i], NULL);
}
```

3.1.5 Distribución Equitativa del Trabajo

Para distribuir las filas de manera equitativa cuando m no es divisible por p:

```
C/C++
int filas_base = m / p;    // División entera

int filas_extra = m % p;   // Filas sobrantes

for (int i = 0; i < p; i++) {
    int filas_este_hilo = filas_base;

    if (i < filas_extra) {
        filas_este_hilo++; // Los primeros hilos reciben una fila extra
    }
}
```

3.1.6 Resultados de Ejecución con 9 Hilos (Grupo 7)

Caso 1: Operación sumar con matriz 18x18 y 9 hilos

None

```
$ ./matrizproc 18 sumar
```

Matriz original (18x18):

```
3 7 2 8 1 9 4 6 5 0 2 4 6 8 1 3 5 7
```

```
8 2 6 1 7 3 9 4 0 5 1 3 5 7 9 2 4 6
```

[... 16 filas más ...]

Procesando con 9 hilos, operación: sumar

Hilo 0 procesando filas 0 a 1

Fila 0: 3 7 2 8 1 9 4 6 5 0 2 4 6 8 1 3 5 7 -> Suma: 85

Fila 1: 8 2 6 1 7 3 9 4 0 5 1 3 5 7 9 2 4 6 -> Suma: 82

Hilo 1 procesando filas 2 a 3

Fila 2: 5 3 7 2 8 4 6 1 9 0 3 5 7 2 4 6 8 1 -> Suma: 81

Fila 3: 2 6 4 8 3 7 1 5 9 2 4 6 8 1 3 5 7 9 -> Suma: 90

[... continúa con los 9 hilos procesando 2 filas cada uno ...]

Procesamiento completado con 9 hilos

Caso 2: Operación max con matriz 9x9 y 9 hilos (1 fila por hilo)

None

```
$ ./matrizproc 9 max
```

Matriz original (9x9):

```
5 2 8 1 6 3 7 4 9
3 9 2 5 1 7 4 6 8
7 4 6 8 2 5 9 1 3
2 8 5 3 9 6 1 7 4
6 1 9 7 4 2 8 3 5
4 7 3 6 8 1 5 9 2
8 5 1 9 3 4 6 2 7
9 3 7 2 5 8 4 1 6
1 6 4 5 7 9 2 8 3
```

Procesando con 9 hilos, operación: max

Hilo 0 procesando filas 0 a 0

Fila 0: 5 2 8 1 6 3 7 4 9 -> Máximo: 9

Hilo 1 procesando filas 1 a 1

Fila 1: 3 9 2 5 1 7 4 6 8 -> Máximo: 9

Hilo 2 procesando filas 2 a 2

Fila 2: 7 4 6 8 2 5 9 1 3 -> Máximo: 9

[... cada hilo procesa exactamente 1 fila ...]

Procesamiento completado con 9 hilos

4. ANÁLISIS COMPARATIVO

4.1 Procesos vs Hilos

Característica	Procesos (fork)	Hilos (pthread)
Creación	~0.5-1 ms	~0.05-0.1 ms
Espacio de memoria	Independiente (copia completa)	Compartido
Comunicación	IPC (pipes, sockets, memoria compartida)	Variables compartidas directamente
Overhead de contexto	Alto (cambio de tabla de páginas)	Bajo (mismo espacio de direcciones)
Aislamiento	Completo (fallo no afecta otros)	Ninguno (fallo puede afectar todos)
Sincronización	Semáforos, señales	Mutex, variables de condición
Uso de recursos	Mayor (memoria duplicada)	Menor (memoria compartida)

Seguridad Mayor (aislamiento) Menor (acceso compartido)

Escalabilidad Limitada por recursos Mejor para muchas tareas

4.2 Análisis Específico para el Grupo 7 (9 Hilos)

Con la configuración de 9 hilos para el Grupo 7, observamos:

1. **Distribución óptima:** Con matrices de tamaño múltiplo de 9 (9, 18, 27, 36...), cada hilo procesa exactamente el mismo número de filas
2. **Distribución con residuo:** Para otros tamaños, los primeros hilos reciben una fila adicional
3. **Rendimiento:** Los 9 hilos muestran excelente paralelización en sistemas con 8+ cores

4.3 Ventajas y Desventajas Observadas

Procesos (matrizsum)

Ventajas:

- Mayor robustez: un fallo en un proceso hijo no afecta a los demás
- Aislamiento natural de datos evita condiciones de carrera
- Facilidad de depuración al poder rastrear cada proceso independientemente
- Mejor para tareas que requieren aislamiento completo

Desventajas:

- Mayor consumo de memoria por la duplicación del espacio de direcciones
- Mayor tiempo de creación y destrucción
- Comunicación más compleja si se requiere compartir resultados
- Overhead significativo con muchos procesos

Hilos (matrizproc con 9 hilos)

Ventajas:

- Creación y destrucción más rápida
- Menor consumo de memoria
- Comunicación directa mediante variables compartidas
- Excelente rendimiento con 9 hilos en procesadores modernos
- Mejor aprovechamiento de múltiples cores

Desventajas:

- Necesidad de sincronización explícita para evitar condiciones de carrera
 - Un error en un hilo puede comprometer todo el proceso
 - Depuración más compleja debido al entrelazado de ejecución
 - Mayor complejidad con 9 hilos concurrentes
-

5. VALIDACIONES Y MANEJO DE ERRORES

5.1 Validaciones Implementadas

Todos los programas incluyen validaciones exhaustivas:

1. Validación de argumentos de entrada

```
C/C++
if (argc != 3) {

    fprintf(stderr, "Uso correcto: %s <tamaño_matriz> <numero_procesos>\n",
    argv[0]);

    return 1;

}
```

2.

Validación de operaciones en matrizproc

```
C/C++
if (strcmp(operacion, "sumar") != 0 && strcmp(operacion, "max") != 0) {

    fprintf(stderr, "Error: Operación inválida \"%s\"\n", operacion);

    return 1;

}
```

3.

Validación de divisibilidad en matrizsum

C/C++

```
if (m % n != 0) {  
    fprintf(stderr, "Error: n (%d) debe ser divisor de m (%d)\n", n, m);  
    return 1;  
}
```

5.2 Manejo de Errores

Los programas implementan manejo robusto de errores:

C/C++

// Manejo de errores en fork()

```
pid_t pid = fork();  
if (pid < 0) {  
    fprintf(stderr, "Error al crear el proceso hijo\n");  
    return 1;  
}
```

// Manejo de errores en pthread_create()

```
if (pthread_create(&threads[i], NULL, procesar_filas, &thread_data[i]) != 0) {  
    fprintf(stderr, "Error al crear el hilo %d\n", i);  
    return 1;  
}
```

```
// Manejo de errores en asignación de memoria

int **matriz = (int **)malloc(m * sizeof(int *));

if (matriz == NULL) {

    fprintf(stderr, "Error al asignar memoria para la matriz\n");

    return 1;

}
```

6. PRUEBAS REALIZADAS

6.1 Plan de Pruebas

Se ejecutaron pruebas sistemáticas para verificar el correcto funcionamiento de todos los programas:

6.2 Resultados de Pruebas Detalladas

Prueba	Comando	Resultado	Estado
COMPILACIÓN			
Compilar matrizsum	gcc -o matrizsum "Parte 1: matrizsum.c"	Compilación exitosa	✓
Compilar ejecutor	gcc -o ejecutor "Parte 1: ejecutor.c"	Compilación exitosa	✓

Pontificia Universidad Javeriana de Bogotá

Compilar matrizproc	<code>gcc -o matrizproc "Parte 2: matrizproc.c" -pthread</code>	Compilación exitosa	✓
------------------------	---	---------------------	---

CASOS DE ÉXITO

Matriz pequeña procesos	<code>./matrizsum 4 2</code>	2 procesos, 2 filas cada uno	✓
----------------------------	------------------------------	---------------------------------	---

Matriz mediana procesos	<code>./matrizsum 12 4</code>	4 procesos, 3 filas cada uno	✓
----------------------------	-------------------------------	---------------------------------	---

Matriz grande procesos	<code>./matrizsum 100 10</code>	10 procesos, 10 filas cada uno	✓
---------------------------	---------------------------------	-----------------------------------	---

Ejecutor básico	<code>./ejecutor 6 3</code>	matrizsum ejecutado correctamente	✓
-----------------	-----------------------------	--------------------------------------	---

Ejecutor grande	<code>./ejecutor 20 5</code>	matrizsum con 20x20 y 5 procesos	✓
-----------------	------------------------------	-------------------------------------	---

Hilos suma pequeña	<code>./matrizproc 9 sumar</code>	9 hilos, 1 fila cada uno	✓
-----------------------	-----------------------------------	--------------------------	---

Hilos suma mediana	<code>./matrizproc 18 sumar</code>	9 hilos, 2 filas cada uno	✓
-----------------------	------------------------------------	------------------------------	---

Hilos suma grande	<code>./matrizproc 27 sumar</code>	9 hilos, 3 filas cada uno	✓
-------------------	------------------------------------	------------------------------	---

Pontificia Universidad Javeriana de Bogotá

Hilos max pequeña	<code>./matrizproc 9 max</code>	9 hilos procesando máximo	✓
-------------------	---------------------------------	---------------------------	---

Hilos max con residuo	<code>./matrizproc 20 max</code>	9 hilos, distribución 3-3-2-2-2-2-2-2	✓
-----------------------	----------------------------------	---------------------------------------	---

CASOS DE ERROR

Argumentos faltantes	<code>./matrizsum 8</code>	Error: uso correcto mostrado	✓
----------------------	----------------------------	------------------------------	---

División no exacta	<code>./matrizsum 10 3</code>	Error: n no divide a m	✓
--------------------	-------------------------------	------------------------	---

Argumentos negativos	<code>./matrizsum -5 2</code>	Error: números positivos	✓
----------------------	-------------------------------	--------------------------	---

Operación inválida	<code>./matrizproc 10 multiplicar</code>	Error: operación no válida	✓
--------------------	--	----------------------------	---

Sin argumentos	<code>./matrizsum</code>	Error: uso correcto	✓
----------------	--------------------------	---------------------	---

6.3 Medición de Rendimiento

Se realizaron mediciones de tiempo con matrices grandes:

```
Shell
# Procesos - Matriz 900x900 con 9 procesos

$ time ./matrizsum 900 9
```

real 0m0.052s

user 0m0.038s

sys 0m0.031s

Hilos - Matriz 900x900 con 9 hilos (Grupo 7)

\$ time ./matrizproc 900 sumar

real 0m0.024s

user 0m0.019s

sys 0m0.005s

Observación: Los hilos con configuración de 9 (Grupo 7) muestran aproximadamente 2.2x mejor rendimiento que los procesos debido al menor overhead.

7. DIFICULTADES ENCONTRADAS Y SOLUCIONES

7.1 Dificultades Técnicas

1. **Problema:** Caracteres especiales mal codificados (ñ, tildes)
 - **Causa:** Codificación de caracteres incorrecta
 - **Solución:** Configurar UTF-8 en el editor y compilador
2. **Problema:** Distribución desigual con 9 hilos cuando m no es múltiplo de 9
 - **Causa:** División entera trunca el resultado
 - **Solución:** Algoritmo de distribución con filas extra para los primeros hilos
3. **Problema:** Compilación con nombres de archivo con espacios
 - **Causa:** Los archivos tienen nombres como "Parte 1: matrizsum.c"
 - **Solución:** Usar comillas en los comandos de compilación

7.2 Lecciones Aprendidas

- La importancia de configurar correctamente el número de hilos según el grupo (9 para Grupo 7)
 - La diferencia práctica entre el modelo de procesos y el de hilos
 - El impacto del número de hilos en el rendimiento
 - La necesidad de validación exhaustiva de entradas
-

8. CONCLUSIONES

8.1 Objetivos Alcanzados

El desarrollo del presente taller permitió alcanzar satisfactoriamente todos los objetivos planteados:

1. **Comprensión de procesos:** Se implementó exitosamente la creación y gestión de procesos mediante `fork()`, entendiendo el modelo de memoria independiente y la necesidad de sincronización mediante `wait()`.
2. **Uso de exec:** Se demostró la capacidad de las llamadas `exec` para reemplazar la imagen de un proceso, permitiendo la ejecución de programas externos.
3. **Implementación con hilos:** Se logró implementar procesamiento paralelo con 9 hilos POSIX (configuración del Grupo 7), aprovechando el modelo de memoria compartida para un acceso eficiente a las estructuras de datos.
4. **Análisis comparativo:** Se identificaron claramente las ventajas y desventajas de cada modelo, comprendiendo cuándo es apropiado usar cada uno.

8.2 Aprendizajes Clave del Grupo 7

- **Configuración específica:** La implementación con 9 hilos (7+2) demostró ser eficiente para el procesamiento paralelo de matrices
- **Distribución de trabajo:** Con 9 hilos, la distribución equitativa funciona perfectamente para matrices de tamaño múltiplo de 9
- **Rendimiento:** Los 9 hilos muestran excelente escalabilidad en sistemas multiprocesador modernos
- **Validación:** La importancia de validar exhaustivamente las entradas para garantizar la robustez del software

8.3 Aplicaciones Prácticas

Los conceptos aprendidos en este taller tienen aplicaciones directas en:

- Desarrollo de servidores web concurrentes
- Procesamiento paralelo de grandes conjuntos de datos
- Implementación de sistemas de tiempo real
- Desarrollo de aplicaciones multiproceso/multihilo
- Optimización de algoritmos computacionalmente intensivos

8.4 Reflexión Final del Grupo 7

Este taller ha proporcionado una base sólida en programación concurrente en sistemas Linux. La configuración específica de 9 hilos para nuestro grupo nos permitió explorar escenarios de paralelización más complejos que grupos con menos hilos. La experiencia práctica obtenida en la implementación de soluciones con procesos e hilos será fundamental para el desarrollo de software de sistemas más complejo.

La comprensión de estos conceptos es esencial para cualquier ingeniero de software que trabaje en desarrollo de sistemas, optimización de rendimiento o computación paralela. El trabajo con 9 hilos concurrentes nos ha dado una perspectiva valiosa sobre los desafíos de la sincronización y coordinación en sistemas con alto nivel de paralelismo.

9. REFERENCIAS

1. Documentación Oficial

- Linux Manual Pages. [man fork](#), [man exec](#), [man pthread_create](#)
- POSIX Threads Programming. Lawrence Livermore National Laboratory.
- GNU C Library Documentation: <https://www.gnu.org/software/libc/manual/>

2. Libros de Referencia

- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Pearson.
- Stevens, W. R., & Rago, S. A. (2013). *Advanced Programming in the UNIX Environment* (3rd ed.). Addison-Wesley.
- Love, R. (2010). *Linux Kernel Development* (3rd ed.). Addison-Wesley.
- Butenhof, D. R. (1997). *Programming with POSIX Threads*. Addison-Wesley.

3. Recursos en Línea

- The Linux Programming Interface: <https://man7.org/tlpi/>
- POSIX Threads Tutorial: <https://computing.llnl.gov/tutorials/pthreads/>
- Linux Process Management: <https://www.kernel.org/doc/html/latest/admin-guide/mm/concepts.html>

4. Documentación del Compilador

- GCC Documentation: <https://gcc.gnu.org/onlinedocs/>
 - GCC Pthread Options: <https://gcc.gnu.org/onlinedocs/gcc/Link-Options.html>
-

ANEXO A: CÓDIGO FUENTE COMPLETO

A.1 Archivo: Parte 1: matrizsum.c

```
C/C++

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>


int main(int argc, char *argv[]) {

    // Validación de argumentos

    if (argc != 3) {

        fprintf(stderr, "Uso correcto: %s <tamaño_matriz> <numero_procesos>\n",
argv[0]);

        fprintf(stderr, "Ejemplo: %s 8 4\n", argv[0]);

        return 1;

    }


    // Conversión y validación de argumentos

    int m = atoi(argv[1]);
```

```
int n = atoi(argv[2]);

if (m <= 0 || n <= 0) {
    fprintf(stderr, "Error: Los argumentos deben ser números positivos\n");
    return 1;
}

// Validar que n sea divisor de m
if (m % n != 0) {
    fprintf(stderr, "Error: n (%d) debe ser divisor de m (%d)\n", n, m);

    fprintf(stderr, "Uso correcto: El número de procesos debe dividir exactamente el tamaño de la matriz\n");

    return 1;
}

// Declarar e inicializar la matriz con valores de ejemplo
int **matriz = (int **)malloc(m * sizeof(int *));

if (matriz == NULL) {
    fprintf(stderr, "Error al asignar memoria para la matriz\n");
    return 1;
}

for (int i = 0; i < m; i++) {
```

```
matriz[i] = (int *)malloc(m * sizeof(int));

if (matriz[i] == NULL) {
    fprintf(stderr, "Error al asignar memoria para la fila %d\n", i);
    return 1;
}

// Inicializar con valores de ejemplo

for (int j = 0; j < m; j++) {
    matriz[i][j] = i + j + 1;
}
}

// Mostrar la matriz original

printf("Matriz original (%dx%d):\n", m, m);

for (int i = 0; i < m; i++) {
    for (int j = 0; j < m; j++) {
        printf("%3d ", matriz[i][j]);
    }
    printf("\n");
}

printf("\n");

// Calcular filas por proceso

int filas_por_proceso = m / n;
```

```
// Crear n procesos hijos

for (int i = 0; i < n; i++) {

    pid_t pid = fork();

    if (pid < 0) {

        fprintf(stderr, "Error al crear el proceso hijo %d\n", i);

        return 1;

    }

    else if (pid == 0) {

        // Código del proceso hijo

        int inicio_fila = i * filas_por_proceso;

        int fin_fila = inicio_fila + filas_por_proceso;

        printf("Proceso hijo PID=%d procesando filas %d a %d\n",

            getpid(), inicio_fila, fin_fila - 1);

        // Procesar las filas asignadas

        for (int fila = inicio_fila; fila < fin_fila; fila++) {

            int suma = 0;

            printf(" Fila %d: ", fila);

            // Mostrar elementos y calcular suma
```

```
    for (int col = 0; col < m; col++) {  
        printf("%d ", matriz[fila][col]);  
        suma += matriz[fila][col];  
    }  
  
    printf("-> Suma = %d (PID: %d)\n", suma, getpid());  
}  
  
// Liberar memoria en el hijo antes de terminar  
for (int j = 0; j < m; j++) {  
    free(matriz[j]);  
}  
free(matriz);  
  
exit(0); // Terminar el proceso hijo  
}  
}  
  
// Proceso padre espera a todos los hijos  
for (int i = 0; i < n; i++) {  
    int status;  
    pid_t pid_hijo = wait(&status);  
    if (pid_hijo < 0) {
```

```
        fprintf(stderr, "Error esperando al proceso hijo\n");
    }
}

printf("\nProceso padre (PID=%d): Todos los hijos han terminado\n", getpid());

// Liberar memoria en el padre
for (int i = 0; i < m; i++) {
    free(matriz[i]);
}
free(matriz);

return 0;
}
```

A.2 Archivo: Parte 1: ejecutor.c

```
C/C++

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>
```

```
int main(int argc, char *argv[]) {

    // Validación de argumentos

    if (argc != 3) {

        fprintf(stderr, "Uso: %s <tamaño_matriz> <numero_procesos>\n", argv[0]);

        fprintf(stderr, "Este programa ejecutará matrizsum con los argumentos proporcionados\n");

        fprintf(stderr, "Ejemplo: %s 8 4\n", argv[0]);

        return 1;

    }

    printf("Ejecutando matrizsum con argumentos: m=%s, n=%s\n\n", argv[1], argv[2]);

    // Crear un proceso hijo para ejecutar matrizsum

    pid_t pid = fork();

    if (pid < 0) {

        fprintf(stderr, "Error al crear el proceso\n");

        return 1;

    }

    else if (pid == 0) {

        // Proceso hijo: ejecutar matrizsum usando execl

        execl("./matrizsum", "matrizsum", argv[1], argv[2], NULL);

    }

}
```



```
// Si llegamos aquí, hubo un error en exec

perror("Error ejecutando matrizsum");

fprintf(stderr, "Asegúrate de que matrizsum esté compilado y en el directorio
actual\n");

exit(1);
}

else {

    // Proceso padre: esperar a que termine el hijo

    int status;

    waitpid(pid, &status, 0);

    if (WIFEXITED(status)) {

        printf("\nEl programa matrizsum terminó con código de salida: %d\n",
            WEXITSTATUS(status));

    }

    else {

        printf("\nEl programa matrizsum terminó de forma anormal\n");

    }

}

return 0;
}
```

A.3 Archivo: Parte 2: matrizproc.c

C/C++

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

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

```
#include <string.h>
```

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

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

```
// Estructura para pasar datos a los hilos
```

```
typedef struct {
```

```
    int **matriz;
```

```
    int m;           // Tamaño de la matriz
```

```
    int inicio_fila; // Primera fila a procesar
```

```
    int fin_fila;    // Última fila a procesar (exclusiva)
```

```
    char operacion[10]; // "sumar" o "max"
```

```
    int thread_id;    // ID del hilo
```

```
} ThreadData;
```

```
// Función que ejecutará cada hilo
```

```
void* procesar_filas(void* arg) {
```

```
    ThreadData* data = (ThreadData*)arg;
```

```
    printf("Hilo %d procesando filas %d a %d\n",
```

```
        data->thread_id, data->inicio_fila, data->fin_fila - 1);
```

```
for (int i = data->inicio_fila; i < data->fin_fila; i++) {  
    printf("Fila %d: ", i);
```

```
    // Mostrar los valores de la fila
```

```
    for (int j = 0; j < data->m; j++) {  
        printf("%3d ", data->matriz[i][j]);  
    }
```

```
    // Realizar la operación solicitada
```

```
    if (strcmp(data->operacion, "sumar") == 0) {  
        int suma = 0;  
        for (int j = 0; j < data->m; j++) {  
            suma += data->matriz[i][j];  
        }  
        printf(" -> Suma: %d\n", suma);  
    }
```

```
    else if (strcmp(data->operacion, "max") == 0) {  
        int max = data->matriz[i][0];  
        for (int j = 1; j < data->m; j++) {  
            if (data->matriz[i][j] > max) {  
                max = data->matriz[i][j];  
            }  
        }
```

```
    }

    printf(" -> Máximo: %d\n", max);

}

}

return NULL;

}

int main(int argc, char *argv[]) {

    // CONFIGURACIÓN PARA GRUPO 7

    int p = 9; // Grupo 7: p = 7 + 2 = 9

    // Validación de argumentos

    if (argc != 3) {

        fprintf(stderr, "Uso: %s <tamaño_matriz> <operacion>\n", argv[0]);

        fprintf(stderr, "Operaciones válidas: sumar, max\n");

        fprintf(stderr, "Ejemplo: %s 10 sumar\n", argv[0]);

        return 1;

    }

    // Obtener y validar el tamaño de la matriz

    int m = atoi(argv[1]);

    if (m <= 0) {
```

```
fprintf(stderr, "Error: El tamaño de la matriz debe ser un número positivo\n");  
  
return 1;  
  
}  
  
// Validar la operación  
  
char* operacion = argv[2];  
  
if (strcmp(operacion, "sumar") != 0 && strcmp(operacion, "max") != 0) {  
    fprintf(stderr, "Error: Operación inválida \"%s\n", operacion);  
  
    fprintf(stderr, "Operaciones válidas: sumar, max\n");  
  
    return 1;  
  
}  
  
// Semilla para números aleatorios  
  
srand(time(NULL));  
  
// Crear la matriz dinámica  
  
int **matriz = (int **)malloc(m * sizeof(int *));  
  
if (matriz == NULL) {  
    fprintf(stderr, "Error al asignar memoria para la matriz\n");  
  
    return 1;  
  
}  
  
for (int i = 0; i < m; i++) {
```

```
matriz[i] = (int *)malloc(m * sizeof(int));

if (matriz[i] == NULL) {
    fprintf(stderr, "Error al asignar memoria para la fila %d\n", i);
    return 1;
}

// Llenar con valores aleatorios (0-9)

for (int j = 0; j < m; j++) {
    matriz[i][j] = rand() % 10;
}
}

// Mostrar la matriz original

printf("Matriz original (%dx%d):\n", m, m);

for (int i = 0; i < m; i++) {
    for (int j = 0; j < m; j++) {
        printf("%3d ", matriz[i][j]);
    }
    printf("\n");
}

printf("\n");

printf("Procesando con %d hilos, operación: %s\n", p, operacion);

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

// Crear arreglos para hilos y datos

```
pthread_t threads[p];
```

```
ThreadData thread_data[p];
```

// Calcular distribución equitativa de filas

```
int filas_base = m / p;
```

```
int filas_extra = m % p;
```

```
int fila_actual = 0;
```

// Crear y lanzar los hilos

```
for (int i = 0; i < p; i++) {
```

```
    thread_data[i].matriz = matriz;
```

```
    thread_data[i].m = m;
```

```
    thread_data[i].inicio_fila = fila_actual;
```

// Distribuir las filas extra entre los primeros hilos

```
int filas_este_hilo = filas_base;
```

```
if (i < filas_extra) {
```

```
    filas_este_hilo++;
```

```
}
```

```
thread_data[i].fin_fila = fila_actual + filas_este_hilo;
```

```
strcpy(thread_data[i].operacion, operacion);

thread_data[i].thread_id = i;

// Crear el hilo

if (pthread_create(&threads[i], NULL, procesar_filas, &thread_data[i]) != 0) {

    fprintf(stderr, "Error al crear el hilo %d\n", i);

    return 1;

}

fila_actual = thread_data[i].fin_fila;

}

// Esperar a que todos los hilos terminen

for (int i = 0; i < p; i++) {

    if (pthread_join(threads[i], NULL) != 0) {

        fprintf(stderr, "Error al esperar el hilo %d\n", i);

    }

}

printf("-----\n");

printf("Procesamiento completado con %d hilos\n", p);

// Liberar memoria
```



```
for (int i = 0; i < m; i++) {  
    free(matriz[i]);  
}  
free(matriz);  
  
return 0;  
}
```

ANEXO B: CAPTURAS DE PANTALLA

B.1 Compilación de los Programas

Captura 1: Compilación de matrizsum

```
Shell  
$ gcc -o matrizsum "Parte 1: matrizsum.c" -Wall
```

Captura 2: Compilación de ejecutor

```
Shell  
$ gcc -o ejecutor "Parte 1: ejecutor.c" -Wall
```

Captura 3: Compilación de matrizproc

Shell

```
$ gcc -o matrizproc "Parte 2: matrizproc.c" -pthread -Wall
```

```
● (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % g  
cc -o matrizsum "Parte 1: matrizsum.c" -Wall  
● (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos %  
gcc -o ejecutor "Parte 1: ejecutor.c" -Wall  
● (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % g  
cc -o matrizproc "Parte 2: matrizproc.c" -pthread -Wall  
○ (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos %
```

B.2 Ejecuciones Exitosas

Captura 4: matrizsum con 8x8 y 4 procesos

Shell

```
$ ./matrizsum 8 4
```

Pontificia Universidad Javeriana de Bogotá

```
● (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % gcc -o matrizsum "Parte 1: matrizsum.c" -Wall
● (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % gcc -o ejecutor "Parte 1: ejecutor.c" -Wall
● (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % gcc -o matrizproc "Parte 2: matrizproc.c" -pthread -Wall
● (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % ./matrizsum 8 4
Matriz original (8x8):
1 2 3 4 5 6 7 8
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9 10
4 5 6 7 8 9 10 11
5 6 7 8 9 10 11 12
6 7 8 9 10 11 12 13
7 8 9 10 11 12 13 14
8 9 10 11 12 13 14 15

Proceso hijo PID=85053 procesando filas 0 a 1
Fila 0: 1 2 3 4 5 6 7 8 -> Suma = 36 (PID: 85053)
Fila 1: 2 3 4 5 6 7 8 9 -> Suma = 44 (PID: 85053)
Proceso hijo PID=85054 procesando filas 2 a 3
Fila 2: 3 4 5 6 7 8 9 10 -> Suma = 52 (PID: 85054)
Fila 3: 4 5 6 7 8 9 10 11 -> Suma = 60 (PID: 85054)
Proceso hijo PID=85055 procesando filas 4 a 5
Fila 4: 5 6 7 8 9 10 11 12 -> Suma = 68 (PID: 85055)
Fila 5: 6 7 8 9 10 11 12 13 -> Suma = 76 (PID: 85055)
Proceso hijo PID=85056 procesando filas 6 a 7
Fila 6: 7 8 9 10 11 12 13 14 -> Suma = 84 (PID: 85056)
Fila 7: 8 9 10 11 12 13 14 15 -> Suma = 92 (PID: 85056)

Proceso padre (PID=85049): Todos los hijos han terminado
○ (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos %
```

Captura 5: matrizsum con 18x18 y 9 procesos

Shell

\$./matrizsum 18 9

Pontificia Universidad Javeriana de Bogotá

```
Matriz original (18x18):
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
 6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35

Proceso hijo PID=85372 procesando filas 0 a 1
  Fila 0: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 -> Suma = 171 (PID: 85372)
  Fila 1: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 -> Suma = 189 (PID: 85372)
Proceso hijo PID=85373 procesando filas 2 a 3
  Fila 2: 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -> Suma = 207 (PID: 85373)
  Fila 3: 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 -> Suma = 225 (PID: 85373)
Proceso hijo PID=85374 procesando filas 4 a 5
  Fila 4: 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 -> Suma = 243 (PID: 85374)
  Fila 5: 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 -> Suma = 261 (PID: 85374)
Proceso hijo PID=85375 procesando filas 6 a 7
  Fila 6: 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 -> Suma = 279 (PID: 85375)
  Fila 7: 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 -> Suma = 297 (PID: 85375)
Proceso hijo PID=85376 procesando filas 8 a 9
  Fila 8: 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 -> Suma = 315 (PID: 85376)
  Fila 9: 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 -> Suma = 333 (PID: 85376)
Proceso hijo PID=85377 procesando filas 10 a 11
Proceso hijo PID=85378 procesando filas 12 a 13
Proceso hijo PID=85379 procesando filas 14 a 15
  Fila 14: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 -> Suma = 423 (PID: 85379)
  Fila 12: 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 -> Suma = 387 (PID: 85378)
  Fila 15: 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 -> Suma = 441 (PID: 85379)
  Fila 13: 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 -> Suma = 405 (PID: 85378)
Proceso hijo PID=85380 procesando filas 16 a 17
  Fila 16: 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 -> Suma = 459 (PID: 85380)
  Fila 17: 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 -> Suma = 477 (PID: 85380)
  Fila 10: 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 -> Suma = 351 (PID: 85377)
  Fila 11: 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 -> Suma = 369 (PID: 85377)

Proceso padre (PID=85371): Todos los hijos han terminado
```

Captura 6: ejecutor con 6x6 y 3 procesos

Shell

\$./ejecutor 6 3

Pontificia Universidad Javeriana de Bogotá

```
● (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % ./matrizsum 18 9
Matriz original (18x18):
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
 6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35

Proceso hijo PID=85372 procesando filas 0 a 1
  Fila 0: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 -> Suma = 171 (PID: 85372)
  Fila 1: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 -> Suma = 189 (PID: 85372)
Proceso hijo PID=85373 procesando filas 2 a 3
  Fila 2: 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -> Suma = 207 (PID: 85373)
  Fila 3: 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 -> Suma = 225 (PID: 85373)
Proceso hijo PID=85374 procesando filas 4 a 5
  Fila 4: 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 -> Suma = 243 (PID: 85374)
  Fila 5: 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 -> Suma = 261 (PID: 85374)
Proceso hijo PID=85375 procesando filas 6 a 7
  Fila 6: 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 -> Suma = 279 (PID: 85375)
  Fila 7: 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 -> Suma = 297 (PID: 85375)
Proceso hijo PID=85376 procesando filas 8 a 9
  Fila 8: 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 -> Suma = 315 (PID: 85376)
  Fila 9: 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 -> Suma = 333 (PID: 85376)
Proceso hijo PID=85377 procesando filas 10 a 11
Proceso hijo PID=85378 procesando filas 12 a 13
Proceso hijo PID=85379 procesando filas 14 a 15
  Fila 14: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 -> Suma = 423 (PID: 85379)
  Fila 12: 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 -> Suma = 387 (PID: 85378)
  Fila 15: 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 -> Suma = 441 (PID: 85379)
  Fila 13: 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 -> Suma = 405 (PID: 85378)
Proceso hijo PID=85380 procesando filas 16 a 17
  Fila 16: 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 -> Suma = 459 (PID: 85380)
  Fila 17: 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 -> Suma = 477 (PID: 85380)
  Fila 10: 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 -> Suma = 351 (PID: 85377)
  Fila 11: 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 -> Suma = 369 (PID: 85377)
```

```
Proceso padre (PID=85371): Todos los hijos han terminado
● (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % ./ejecutor 6 3
Ejecutando matrizsum con argumentos: m=6, n=3

Matriz original (6x6):
 1  2  3  4  5  6
 2  3  4  5  6  7
 3  4  5  6  7  8
 4  5  6  7  8  9
 5  6  7  8  9 10
 6  7  8  9 10 11

Proceso hijo PID=85684 procesando filas 0 a 1
  Fila 0: 1 2 3 4 5 6 -> Suma = 21 (PID: 85684)
  Fila 1: 2 3 4 5 6 7 -> Suma = 27 (PID: 85684)
Proceso hijo PID=85685 procesando filas 2 a 3
  Fila 2: 3 4 5 6 7 8 -> Suma = 33 (PID: 85685)
  Fila 3: 4 5 6 7 8 9 -> Suma = 39 (PID: 85685)
Proceso hijo PID=85686 procesando filas 4 a 5
  Fila 4: 5 6 7 8 9 10 -> Suma = 45 (PID: 85686)
  Fila 5: 6 7 8 9 10 11 -> Suma = 51 (PID: 85686)

Proceso padre (PID=85683): Todos los hijos han terminado

El programa matrizsum terminó con código de salida: 0
○ (base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos %
```

Captura 7: matrizproc con 9x9 y operación sumar (9 hilos)

Shell

```
$ ./matrizproc 9 sumar
```

```
(base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % ./matrizproc 9 sumar
Matriz original (9x9):
 2  0  2  7  3  8  0  0  8
 7  3  7  8  4  6  6  5  5
 8  9  4  1  6  2  3  2  3
 3  1  2  9  5  7  4  3  6
 6  0  1  7  4  4  0  6  6
 3  2  7  6  3  8  0  0  3
 9  5  3  7  8  3  5  3  6
 9  9  1  6  5  5  8  3  0
 6  6  1  2  9  8  8  7  8

Procesando con 9 hilos, operación: sumar
-----
Hilo 0 procesando filas 0 a 0
Fila 0:  2  0  2  7  3  8  0  0  8  -> Suma: 30
Hilo 1 procesando filas 1 a 1
Fila 1:  7  3  7  8  4  6  6  5  5  -> Suma: 51
Hilo 3 procesando filas 3 a 3
Fila 3:  3  1  2  9  5  7  4  3  6  -> Suma: 40
Hilo 4 procesando filas 4 a 4
Fila 4:  6  0  1  7  4  4  0  6  6  -> Suma: 34
Hilo 2 procesando filas 2 a 2
Fila 2:  8  9  4  1  6  2  3  2  3  -> Suma: 38
Hilo 5 procesando filas 5 a 5
Fila 5:  3  2  7  6  3  8  0  0  3  -> Suma: 32
Hilo 7 procesando filas 7 a 7
Fila 7:  9  9  1  6  5  5  8  3  0
Hilo 6 procesando filas 6 a 6
Fila 6:  9  5  3  7  8  3  5  3  6  -> Suma: 49
-> Suma: 46
Hilo 8 procesando filas 8 a 8
Fila 8:  6  6  1  2  9  8  8  7  8  -> Suma: 55
-----
Procesamiento completado con 9 hilos
(base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos %
```

Captura 8: matrizproc con 18x18 y operación sumar (9 hilos)

Shell

```
$ ./matrizproc 18 sumar
```

Pontificia Universidad Javeriana de Bogotá

```
Matriz original (18x18):
1 5 6 9 5 0 4 3 6 5 3 4 8 1 8 9 1 1
1 9 8 8 7 6 2 8 5 3 6 4 0 0 2 9 3 4
0 3 5 1 4 9 8 1 5 4 9 6 5 5 8 0 7 2
5 0 3 4 0 7 4 7 1 1 7 4 6 7 9 5 1 7
5 9 8 9 5 3 3 3 9 4 7 8 5 3 5 1 8 5
8 0 8 2 4 0 6 3 2 7 2 8 1 4 7 1 4 0
5 7 0 2 2 2 6 7 1 2 2 5 8 1 9 9 9 9
2 8 6 8 4 5 8 7 5 0 9 6 3 7 8 8 8 2
7 3 8 0 5 6 8 4 9 0 7 5 8 9 9 0 3 0
5 2 1 1 3 5 7 4 6 4 8 4 7 5 7 2 9 3
9 3 6 5 0 3 5 0 8 2 6 6 9 4 3 9 7 2
4 4 3 0 8 5 3 2 9 7 1 6 4 2 0 5 1 6
2 0 4 5 0 7 7 0 4 4 8 7 7 5 7 1 1 2
8 8 2 9 8 5 7 3 7 8 3 0 2 4 3 9 9 3
5 4 3 0 2 4 9 6 9 8 4 0 6 7 5 9 7 5
8 0 1 5 3 7 6 4 1 2 6 0 4 8 3 3 1 3
9 9 8 5 6 9 1 2 1 3 5 6 8 7 8 5 6 4
6 3 9 9 3 8 8 4 0 2 4 0 4 1 3 7 6 3

Procesando con 9 hilos, operación: sumar
-----
Hilo 0 procesando filas 0 a 1
Fila 0: 1 5 6 9 5 0 4 3 6 5 3 4 8 1 8 9 1 1 -> Suma: 79
Fila 1: 1 9 8 8 7 6 2 8 5 3 6 4 0 0 2 9 3 4 -> Suma: 85
Hilo 2 procesando filas 4 a 5
Fila 4: 5 9 8 9 Hilo 7 procesando filas 14 a 15
Fila 14: 5 4 Hilo 5 procesando filas 10 a 11
Fila 10: Hilo 4 procesando filas 8 a 9
9 Fila 8: 3 6 5 0 Hilo 6 procesando filas 12 a 13
Fila 12: Hilo 3 procesando filas 6 a 7
3 Fila 6: 0 7 5 7 0 2 2 4 9 6 9 8 4 0 6 5 3 3 3 9 4 7 8 5 3 5 1 8 5 -> Suma: 100
Fila 5: 8 0 8 2 4 0 6 3 2 7 2 8 1 4 7 1 4 0 -> Suma: 67
3 8 0 5 6 8 4 9 0 7 5 8 9 9 0 3 0 -> Suma: 91
Fila 9: 5 2 1 1 3 5 7 4 6 2 2 6 7 1 2 2 5 8 1 9 9 9 9 Hilo 8 procesando filas 16 a 17
Fila 16: 9 3 -> Suma: 86
Fila 7: 2 8 6 8 4 5 8 7 5 0 9 6 3 7 8 8 8 2 -> Suma: 104
2 9 8 5 6 9 1 2 1 3 5 6 8 7 8 5 6 4 -> Suma: 102
Fila 17: 6 3 9 9 3 8 8 4 0 2 4 0 4 1 3 7 4 8 5 0 8 2 6 6 9 4 3 9 7 2 -> Suma: 87
Fila 11: 4 4 3 0 8 5 3 2 9 7 1 6 4 2 0 5 1 6 -> Suma: 70
0 4 5 0 7 7 0 4 4 8 7 7 5 9 7 5 -> Suma: 93
Fila 15: 8 0 1 5 3 7 6 4 1 4 Hilo 1 procesando filas 2 a 3
7 5 7 2 9 3 -> Suma: 83
Fila 2: 0 3 5 1 4 9 8 1 7 5 7 1 1 2 -> Suma: 71
Fila 13: 8 8 2 9 8 5 7 3 7 8 3 0 2 4 3 9 9 3 -> Suma: 98
2 6 0 6 3 5 4 9 4 8 3 3 1 3 -> Suma: 65
6 5 5 8 0 7 2 -> Suma: 82
Fila 3: 5 0 3 4 0 7 4 7 1 1 7 4 6 7 9 5 1 7 -> Suma: 78
-> Suma: 80
```

Captura 9: matrizproc con 9x9 y operación max (9 hilos)

Shell

```
$ ./matrizproc 9 max
```

```
(base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % ./matrizproc 9 max
Matriz original (9x9):
4 7 5 2 1 7 6 5 3
1 2 9 3 5 0 9 1 2
7 9 0 2 3 4 2 8 1
2 5 2 5 9 8 8 2 8
2 9 3 6 9 1 4 7 4
1 5 1 9 6 2 1 1 6
0 7 3 7 2 6 1 8 2
4 0 8 7 7 9 4 0 5
8 8 6 6 0 6 3 2 1

Procesando con 9 hilos, operación: max
-----
Hilo 0 procesando filas 0 a 0
Fila 0: 4 7 5 2 1 7 6 5 3 Hilo 2 procesando filas 2 a 2
Hilo 3 procesando filas 3 a 3
Hilo 4 procesando filas 4 a 4
-> Máximo: 7
Hilo 5 procesando filas 5 a 5
Fila 4: 2 9 3 6 9 1 4 7 4 Fila 5: Hilo 6 procesando filas 6 a 6
Hilo 7 procesando filas 7 a 7
Fila 7: 4 0 8 7 7 9 4 0 5 -> Máximo: 9
Fila 2: 7 9 0 2 3 4 1 5 1 9 6 2 1 1 6 -> Máximo: 9
Fila 6: 0 7 3 7 2 6 1 8 2 -> Máximo: 8
2 8 1 -> Máximo: 9
Hilo 1 procesando filas 1 a 1
Fila 1: 1 2 9 3 5 0 9 1 2 -> Máximo: 9
Hilo 8 procesando filas 8 a 8
Fila 8: 8 8 6 6 0 6 3 2 1 -> Máximo: 8
Fila 3: 2 5 2 5 9 8 8 2 8 -> Máximo: 9
-> Máximo: 9
-----
Procesamiento completado con 9 hilos
(base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos %
```

B.3 Casos de Error

Captura 10: Error - Argumentos faltantes

Shell

```
$ ./matrizsum 8
```

```
(base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % ./matrizsum 8
Uso correcto: ./matrizsum <tamaño_matriz> <numero_procesos>
Ejemplo: ./matrizsum 8 4
(base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos %
```

Captura 11: Error - n no divide a m

Shell

```
$ ./matrizsum 10 3
```

```
(base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos % ./matrizsum 10 3
Error: n (3) debe ser divisor de m (10)
Uso correcto: El número de procesos debe dividir exactamente el tamaño de la matriz
(base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos %
```

Captura 12: Error - Operación inválida

Shell

```
$ ./matrizproc 10 multiplicar
```

```
$ ./matrizproc 10 multiplicar
```

```
Shell
$ time ./matrizsum 900 9
```

```
$ time ./matrizsum 900 9
```

```
Shell
$ time ./matrizproc 900 sumar
```

```
$ time ./matrizproc 900 sumar
```

Pontificia Universidad Javeriana de Bogotá

```
> Suma: 4051
Fila 198: 0 6 4 9 1 7 4 2 2 9 8 1 0 3 8 1 7 6 5 1 8 9 4 0 0 9 8 0 2 5 3 6 2 9 8 6 3 7
1 8 6 3 0 5 8 5 2 4 0 2 0 9 8 6 5 4 0 6 8 3 4 4 2 9 2 2 5 3 8 5 0 1 0 0 9 2 4 5 0 7 3
7 8 1 0 9 9 6 4 1 9 1 9 8 4 8 4 4 9 6 4 2 2 5 0 1 3 6 7 0 3 7 5 9 2 5 4 8 3 0 5 0 0 1
5 0 8 5 6 5 9 2 0 9 7 0 8 7 4 4 2 4 0 6 1 2 0 9 6 7 5 1 0 6 8 3 5 6 1 4 7 7 5 9 5 0 5
8 6 8 3 1 6 7 9 1 7 6 4 9 7 2 7 5 2 1 8 7 8 2 8 4 9 3 4 1 9 2 2 7 3 5 2 8 5 8 3 7 2 5
3 4 7 3 7 6 5 9 8 3 7 3 6 2 2 1 2 7 8 2 0 8 5 6 0 8 6 5 2 5 3 5 6 2 8 1 5 1 7 5 2 4 7
5 5 0 8 6 3 6 2 1 1 2 5 5 3 9 2 8 2 0 6 9 6 8 7 7 2 8 4 6 2 5 9 9 6 2 2 7 9 8 3 4 6 9 4
1 4 3 0 3 0 5 3 6 7 7 1 5 8 5 9 2 5 0 1 4 8 8 0 0 3 1 7 0 5 1 0 5 8 2 2 5 7 8 1 5 7 4
0 9 4 1 5 9 5 3 5 6 6 0 1 2 3 7 6 1 3 1 4 0 7 7 7 5 1 2 5 7 4 7 4 8 4 7 6 7 5 0 3 8 6
4 2 0 1 8 4 9 2 9 2 5 2 4 7 8 1 7 3 2 9 6 3 3 8 2 2 1 0 7 9 3 7 7 5 5 2 0 6 9 5 7 9
8 3 4 2 3 0 6 4 9 5 7 6 2 9 7 2 7 9 4 9 2 5 5 8 9 6 3 3 3 1 6 2 8 8 3 6 8 9 4 5 0 3 9
1 9 3 0 3 8 7 0 7 4 0 3 4 1 8 0 3 4 9 8 8 2 7 1 5 8 9 1 6 0 9 4 2 4 3 5 3 4 8 8 0 4 3
5 9 7 9 2 7 8 0 6 8 1 3 3 6 3 2 8 2 6 1 0 9 3 1 1 1 0 8 9 9 0 9 9 6 4 5 7 6 3 2 5 9 6
4 4 6 7 7 6 6 8 9 2 7 9 5 4 6 0 6 8 6 6 3 2 4 2 9 2 6 9 6 7 7 1 7 4 0 2 9 5 0 8 0 5 4
8 9 7 1 9 8 7 7 0 0 4 0 1 0 3 8 8 1 9 0 1 1 0 8 9 9 5 9 8 7 2 6 7 4 4 4 0 1 3 9 2 7 3
6 2 5 9 3 5 6 9 7 8 2 4 0 1 2 9 4 1 7 6 6 5 5 4 8 6 4 5 0 3 5 1 7 6 3 8 2 0 0 2 8 6 7
8 9 7 7 2 1 8 1 9 7 2 1 7 0 0 1 8 2 5 6 1 2 7 7 2 2 5 5 4 6 9 4 7 7 7 0 9 0 2 1 3 8 2
7 6 7 5 9 1 8 2 8 7 6 7 9 7 2 7 1 5 5 9 3 1 0 2 8 1 3 2 1 6 9 8 2 0 0 4 2 0 9 8 4 5 2
4 5 6 2 6 4 0 1 0 4 1 3 4 6 8 1 1 9 3 6 7 1 8 2 7 8 1 3 4 9 4 9 3 7 9 1 9 9 0 8 6 4
4 3 1 4 1 7 8 0 7 1 8 5 7 8 1 5 8 5 7 2 0 9 8 6 4 4 3 0 3 4 8 2 7 0 6 9 6 0 2 5 8 6 6
4 1 0 6 3 9 3 5 4 8 6 0 1 7 6 6 6 4 3 9 8 9 7 1 2 2 1 3 9 5 6 7 5 2 5 1 3 0 9 9 1 8 7
> Suma: 4143
Fila 199: 4 0 4 9 6 5 7 0 2 6 2 0 1 8 5 8 0 5 0 2 2 4 6 1 8 4 2 9 4 4 0 2 4 2 2 1 0 6 5 8
7 0 0 0 2 1 8 0 4 8 4 0 7 7 0 7 9 4 2 0 1 0 2 7 1 4 3 3 1 0 6 5 9 5 7 1 4 1 9 2 6 9 1
7 7 4 4 8 1 9 7 5 0 8 9 9 3 6 9 7 4 4 7 6 7 1 6 6 6 4 0 6 7 7 5 8 3 1 9 0 5 1 1 6 7 8
8 3 2 9 1 3 7 0 2 8 0 1 9 6 0 8 0 0 6 3 8 3 0 6 2 3 4 4 2 6 9 4 7 8 0 4 2 4 1 2 1 2 9
7 0 1 3 6 9 1 7 7 0 1 7 5 6 2 4 8 3 1 4 0 8 0 3 8 5 0 3 0 8 2 5 9 3 3 0 3 8 8 5 1 5 5
4 8 5 6 5 9 8 5 0 2 4 2 6 2 9 9 6 3 0 1 5 7 3 6 1 1 1 3 4 4 3 3 7 3 8 8 2 5 8 1 1 1 2
0 8 6 5 9 5 0 3 9 2 4 6 4 9 7 3 7 2 7 4 6 9 7 8 5 1 6 3 8 8 4 8 5 7 1 1 9 0 9 4 0 1 0
0 8 6 1 2 8 8 8 0 9 4 8 1 4 3 8 7 9 8 3 5 7 6 1 8 6 0 1 0 8 9 2 2 1 2 4 9 8 5 0 3 2 1
7 1 5 4 5 3 1 6 4 5 0 6 8 8 4 2 0 8 9 5 8 6 2 9 0 4 9 8 9 2 7 6 0 5 0 2 3 7 6 7 6 3 5
8 1 6 1 2 5 6 3 6 6 8 7 1 8 3 1 5 8 9 4 7 0 8 9 0 2 6 5 6 4 4 9 6 4 8 3 4 6 4 9 0 7 0
8 4 9 8 5 4 5 2 3 7 7 2 4 7 3 0 5 1 9 4 6 8 0 9 1 4 1 2 2 2 0 4 6 7 9 7 5 4 3 1 7 2 3
5 3 1 9 0 3 0 5 0 0 1 8 7 0 8 6 6 4 9 0 5 4 2 0 9 5 5 1 0 7 9 3 6 0 8 1 4 4 7 4 1 5 4
7 1 6 1 6 6 1 7 7 1 7 1 5 3 7 6 0 6 0 0 9 9 3 9 0 6 3 7 7 5 8 8 6 6 6 0 1 6 0 7 3 2 7
3 0 3 0 1 1 5 0 2 7 9 3 7 4 5 3 3 8 2 5 1 0 1 1 0 4 7 8 9 5 3 4 4 3 3 5 2 6 0 7 5 3 4
6 8 3 0 7 5 9 2 5 4 6 1 9 8 1 2 0 4 7 7 9 9 4 0 4 0 1 1 9 0 6 9 1 9 6 7 0 3 3 7 7 1 7 0
4 3 5 8 4 3 8 5 4 5 4 8 9 4 0 4 5 6 8 9 5 1 3 1 1 2 7 0 9 6 7 5 9 5 3 6 1 0 3 6 8 4 9
7 2 7 6 8 7 4 4 9 1 2 3 9 0 3 9 8 2 6 3 0 4 8 6 0 5 0 3 7 0 2 0 1 5 3 1 1 0 2 3 0 5 6
4 9 7 5 1 4 4 7 6 1 5 9 3 7 3 7 4 5 0 5 7 8 8 3 8 7 7 7 9 1 1 9 9 1 8 4 6 3 4 9 4 4 4
4 4 3 7 6 8 5 1 0 3 1 3 0 2 4 0 4 5 7 0 8 2 5 0 9 5 0 0 7 6 5 2 6 0 3 7 7 3 7 1 8 4
7 0 1 2 4 5 8 2 0 0 3 8 8 9 5 4 5 0 6 0 2 5 8 5 2 8 7 1 4 2 2 5 9 5 8 1 9 2 2 3 6 4 7
6 4 2 3 9 4 1 3 7 3 7 1 9 1 2 0 4 0 9 3 6 3 9 3 5 8 4 6 7 2 8 9 7 3 7 0 3 1 9 6 6 4 5
> Suma: 3919
-----
Procesamiento completado con 9 hilos
./matrizproc 900 sumar 0.34s user 1.50s system 61% cpu 2.969 total
p(base) santosa@MacBook-Pro-de-Santos-1772 T12_Procesos_Hilos %
```

ANEXO C: INSTRUCCIONES DE COMPILACIÓN Y EJECUCIÓN

C.1 Requisitos Previos

Shell

Verificar instalación de GCC

```
gcc --version
```

Instalar herramientas de desarrollo si es necesario

```
sudo apt-get update
```

```
sudo apt-get install build-essential
```

C.2 Compilación

Shell

Navegar al directorio del proyecto

cd T12_Procesos_Hilos

Compilar todos los programas

gcc -o matrizsum "Parte 1: matrizsum.c" -Wall

gcc -o ejecutor "Parte 1: ejecutor.c" -Wall

gcc -o matrizproc "Parte 2: matrizproc.c" -pthread -Wall

Verificar que se crearon los ejecutables

ls -la matrizsum ejecutor matrizproc

C.3 Ejecución

Programa matrizsum

Shell

Sintaxis

./matrizsum <tamaño_matriz> <numero_procesos>

Ejemplos

./matrizsum 8 4 # Matriz 8x8 con 4 procesos

./matrizsum 18 9 # Matriz 18x18 con 9 procesos

./matrizsum 12 3 # Matriz 12x12 con 3 procesos

Programa ejecutor

Shell

Sintaxis

```
./ejecutor <tamaño_matriz> <numero_procesos>
```

Ejemplos

```
./ejecutor 6 3 # Ejecuta matrizsum con 6x6 y 3 procesos
```

```
./ejecutor 12 4 # Ejecuta matrizsum con 12x12 y 4 procesos
```

Programa matrizproc (9 hilos - Grupo 7)

Shell

Sintaxis

```
./matrizproc <tamaño_matriz> <operacion>
```

Ejemplos con operación sumar

```
./matrizproc 9 sumar # 9x9 con 9 hilos (1 fila por hilo)
```

```
./matrizproc 18 sumar # 18x18 con 9 hilos (2 filas por hilo)
```

```
./matrizproc 27 sumar # 27x27 con 9 hilos (3 filas por hilo)
```

Ejemplos con operación max

```
./matrizproc 9 max # Encuentra máximo de cada fila
```

```
./matrizproc 20 max # Con distribución no equitativa
```

C.4 Scripts de Prueba

Shell

Dar permisos de ejecución a los scripts

```
chmod +x ejecutar_pruebas.sh pruebas_rapidas.sh
```

Ejecutar pruebas rápidas

```
./pruebas_rapidas.sh
```

Ejecutar todas las pruebas con pausas para capturas

```
./ejecutar_pruebas.sh
```

ANEXO D: ANÁLISIS DE COMPLEJIDAD

D.1 Complejidad Temporal

Programa	Complejidad	Descripción
matrizsum	$O(m^2/n)$ por proceso	Cada proceso suma m^2/n elementos
ejecutor	$O(1)$ + tiempo de matrizsum	Solo crea proceso y ejecuta
matrizproc	$O(m^2/p)$ por hilo	Cada hilo procesa m^2/p elementos

D.2 Complejidad Espacial

Programa	Memoria	Descripción
matrizsum	$O(m^2 \times n)$	n copias de la matriz
ejecutor	$O(1)$	Solo variables locales
matrizproc	$O(m^2 + p)$	Una matriz + p estructuras ThreadData