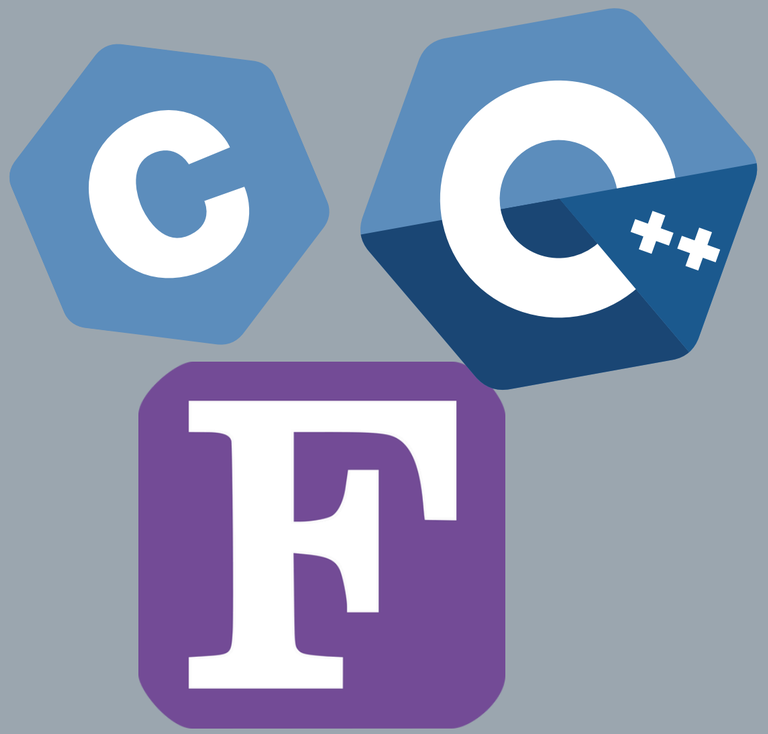
OpenMP desde la terminal:
cómo el kernel ejecuta el paralelismo.

Sistemas operativos

Sierra García Mariana
Tapia García Andres

Contenido

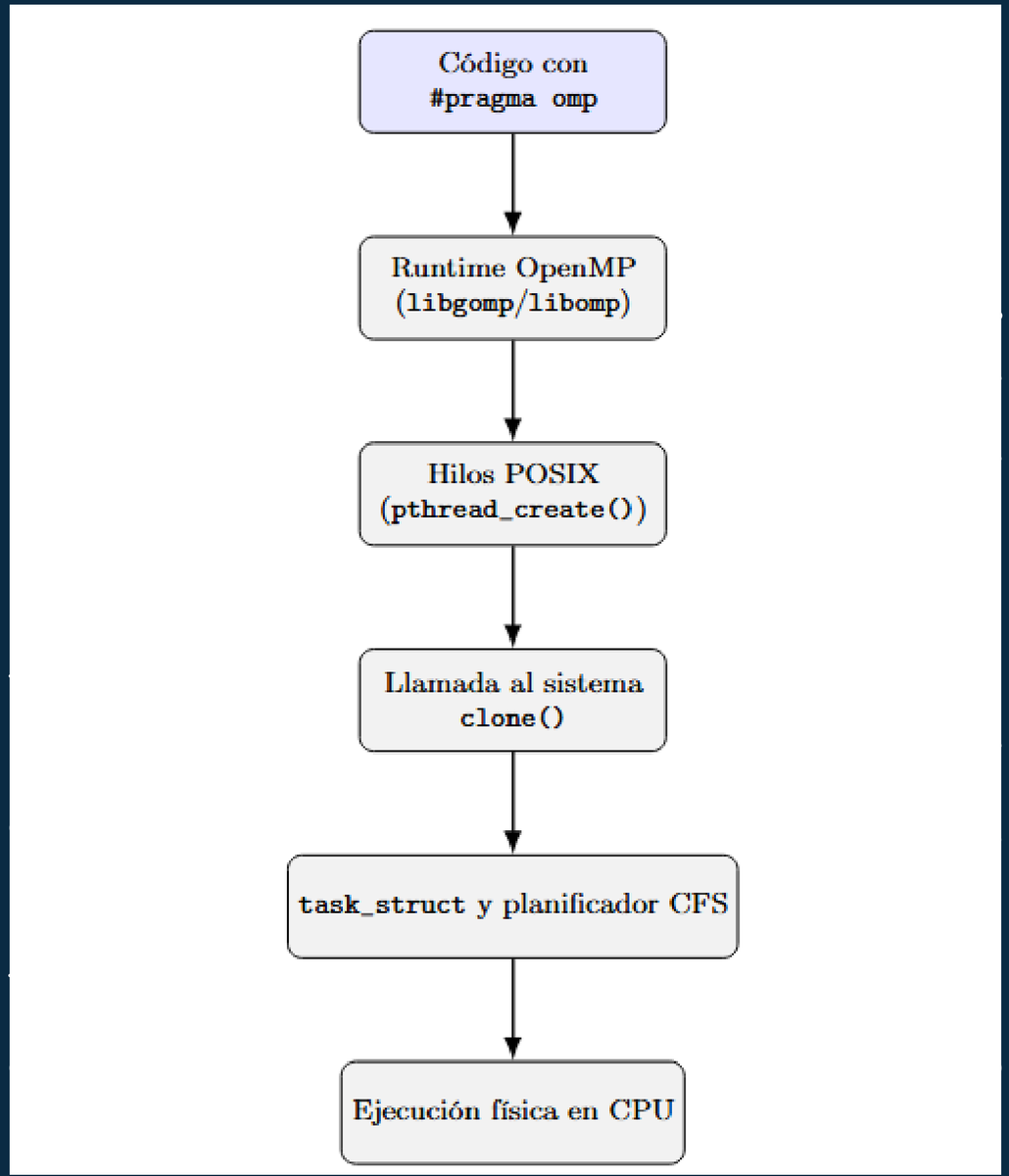
OpenMP



Es una API estándar para paralelismo en memoria compartida que permite crear y coordinar hilos mediante directivas de compilador (#pragma). Surgió en los años 90 y simplifica el paralelismo en C, C++ y Fortran sin necesidad de usar directamente pthreads.

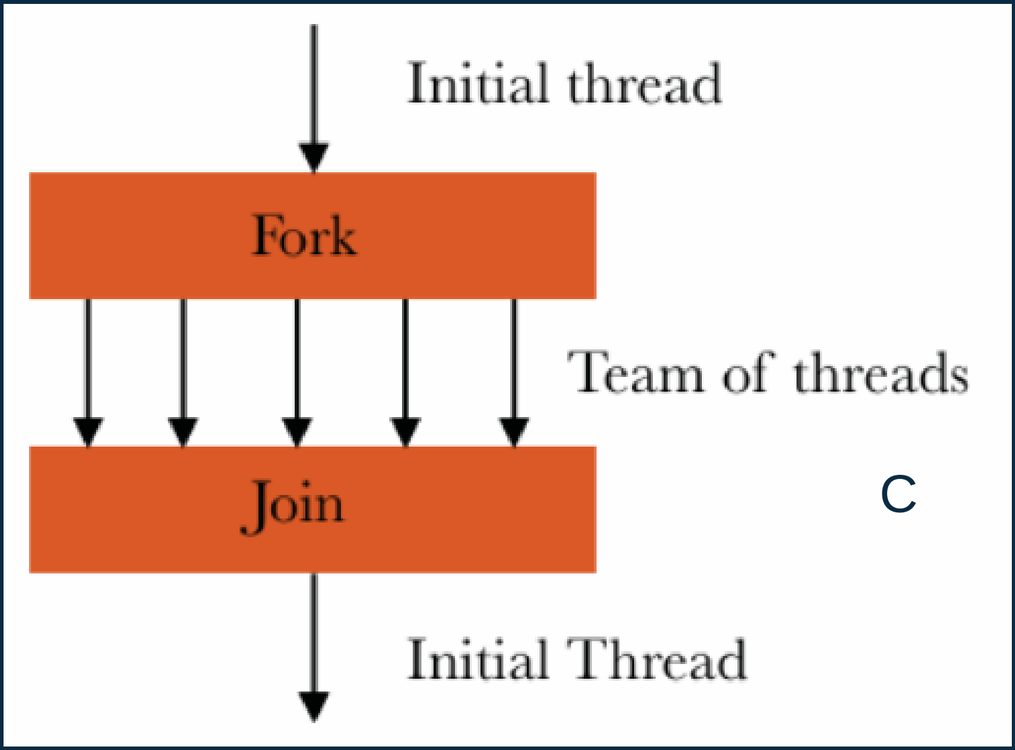
Cómo Kernel ejecuta hilos

Linux trata procesos y hilos de la misma forma mediante la estructura task_struct. Los hilos se crean mediante la llamada al sistema clone(), y su planificación está a cargo del Completely Fair Scheduler (CFS), que distribuye equitativamente el tiempo de CPU.



Fork-Join

El hilo master crea un equipo de hilos (fork) para ejecutar una región paralela, y luego todos terminan y se sincronizan antes de continuar la ejecución secuencial (join). No usa fork() porque todos los hilos comparten memoria.



C

Fork: OpenMP crea hilos usando pthreads → clone() → tareas del kernel.
Join: El master espera; los hilos terminan y CFS los finaliza ordenadamente.

Ejemplo

Ejemplo de manejo de Hilos en OpenMP con clausulas

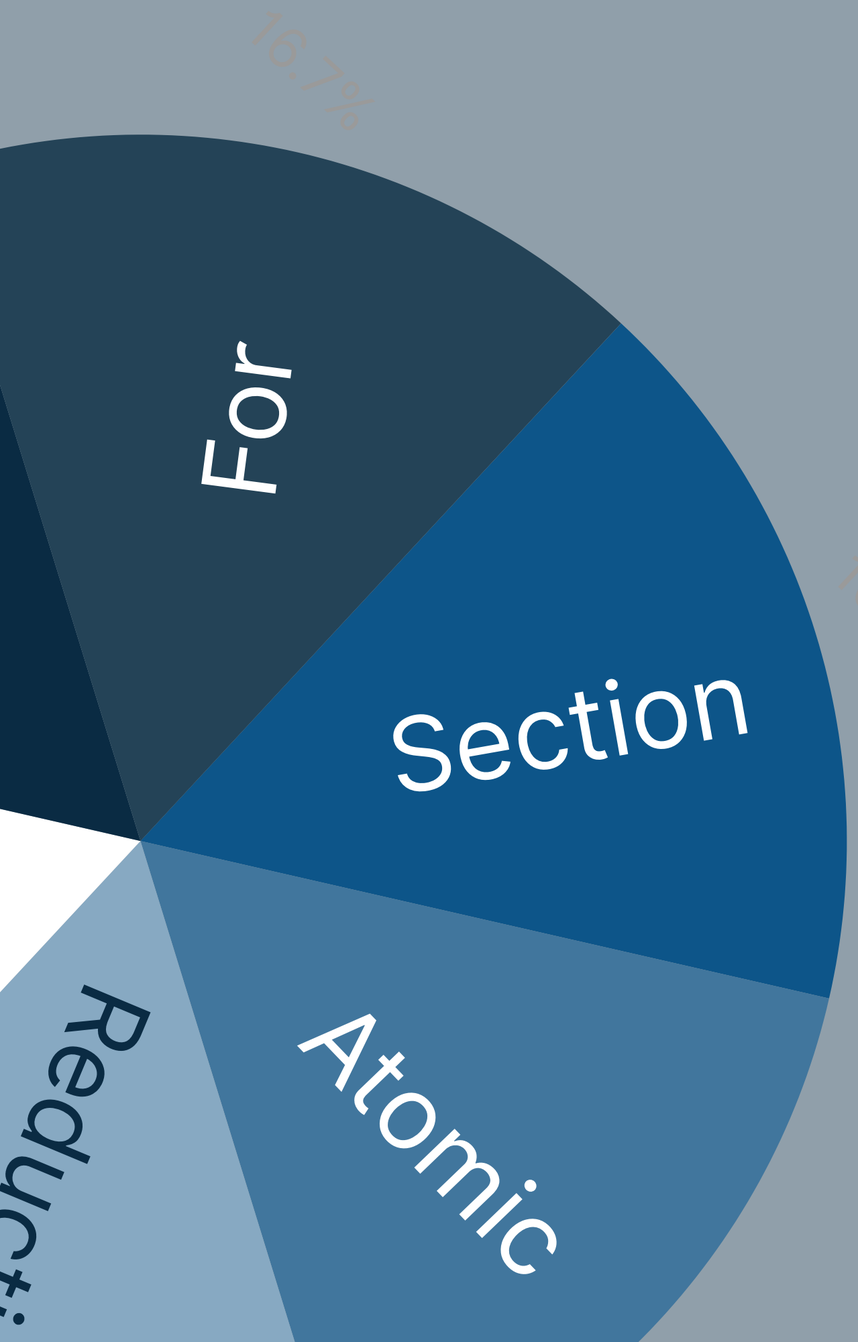
```c
#include <stdio.h>
#include <omp.h>

int main() {
  #pragma omp parallel num_threads(4)
  {
    printf("Hilo %d de %d\\n",
           omp_get_thread_num(),
           omp_get_num_threads());
  }
}
```

```
s/SierraMariana-TapiaAndres/Programas$ gcc -fopenmp ManejoHilos.c -o test.out
s/SierraMariana-TapiaAndres/Programas$ ./test.out
```

```
Hilo 0 de 4
Hilo 2 de 4
Hilo 3 de 4
Hilo 1 de 4
```

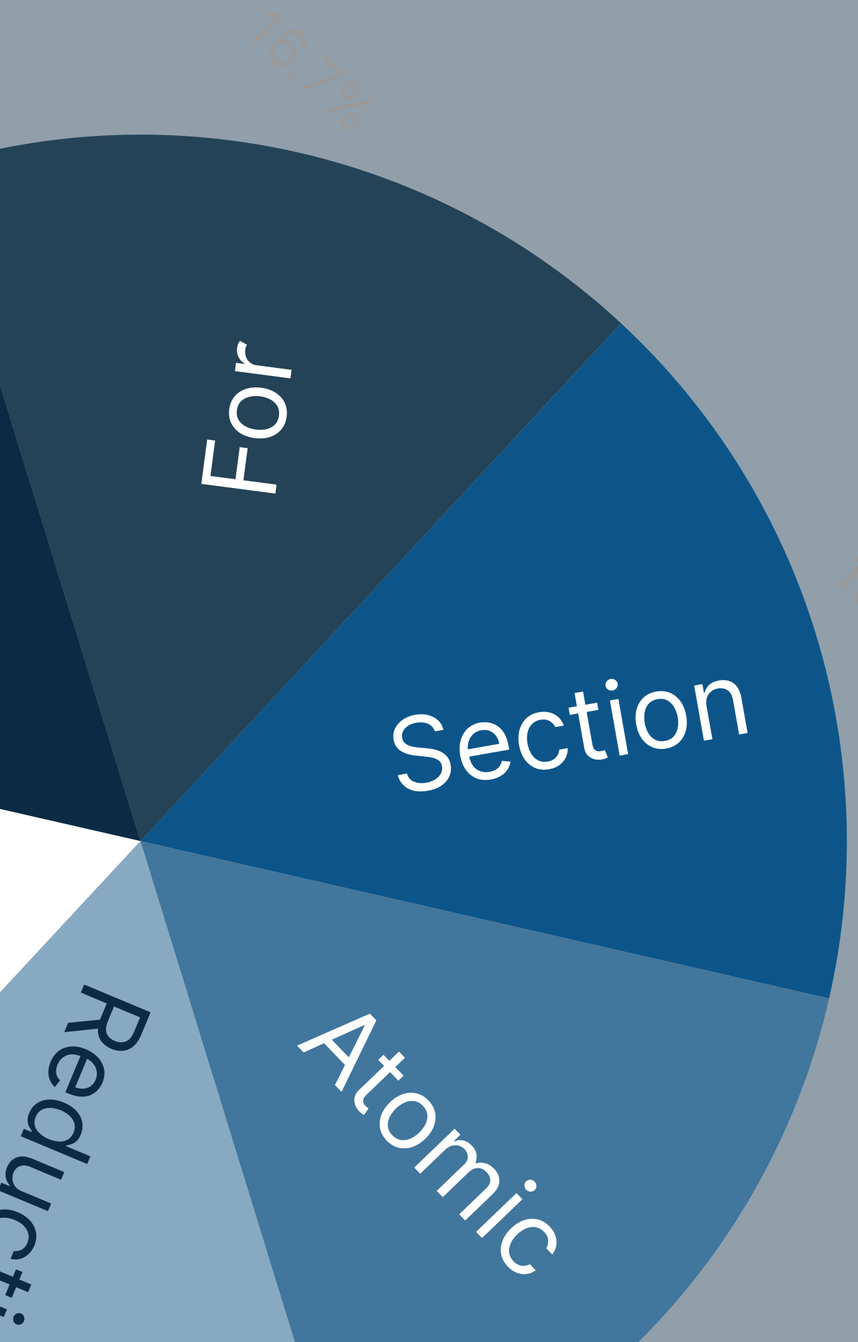
```
Hilo 2 de 4
Hilo 0 de 4
Hilo 3 de 4
Hilo 1 de 4
```

Clasulas y sincronización



Ofrece diferentes formas de trabajar con los hilos, mediante sus clausulas.

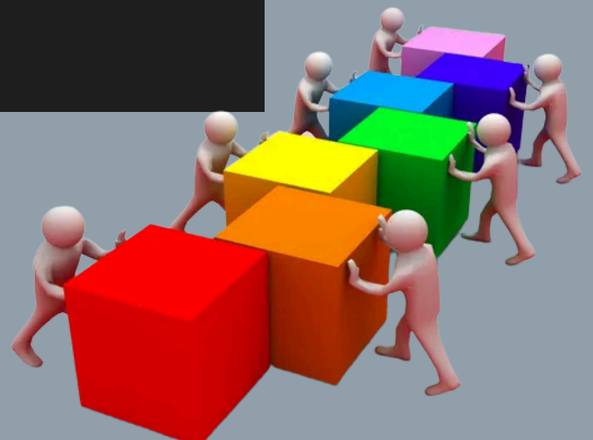
Clasulas y sincronización



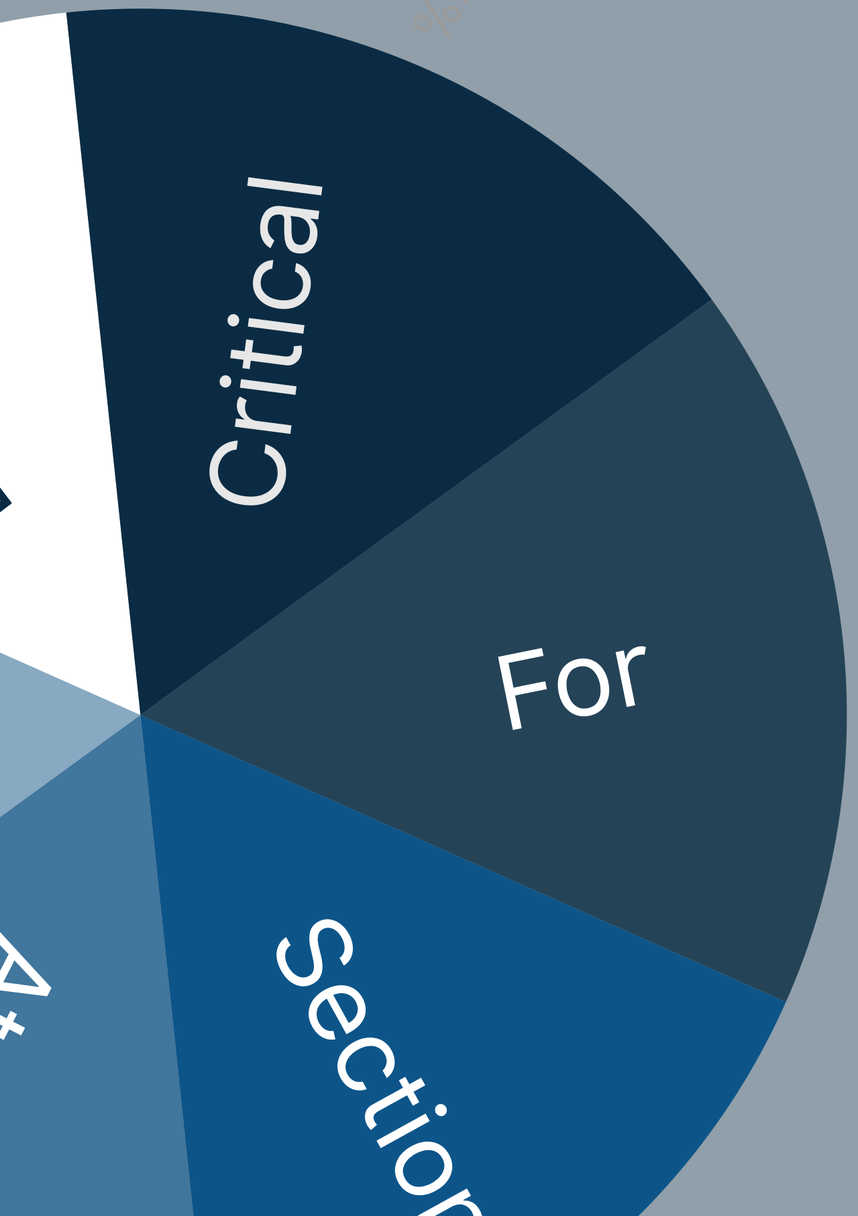

```c
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            printf("Sección 1 ejecutada por el hilo %d\n",
                omp_get_thread_num());
        }

        #pragma omp section
        {
            printf("Sección 2 ejecutada por el hilo %d\n",
                omp_get_thread_num());
        }
    }
}
```

```
s/Programas$ gcc -fopenmp section.c -o test.out
s/Programas$ ./test.out
```

```
Sección 1 ejecutada por el hilo 3
Sección 2 ejecutada por el hilo 2
```



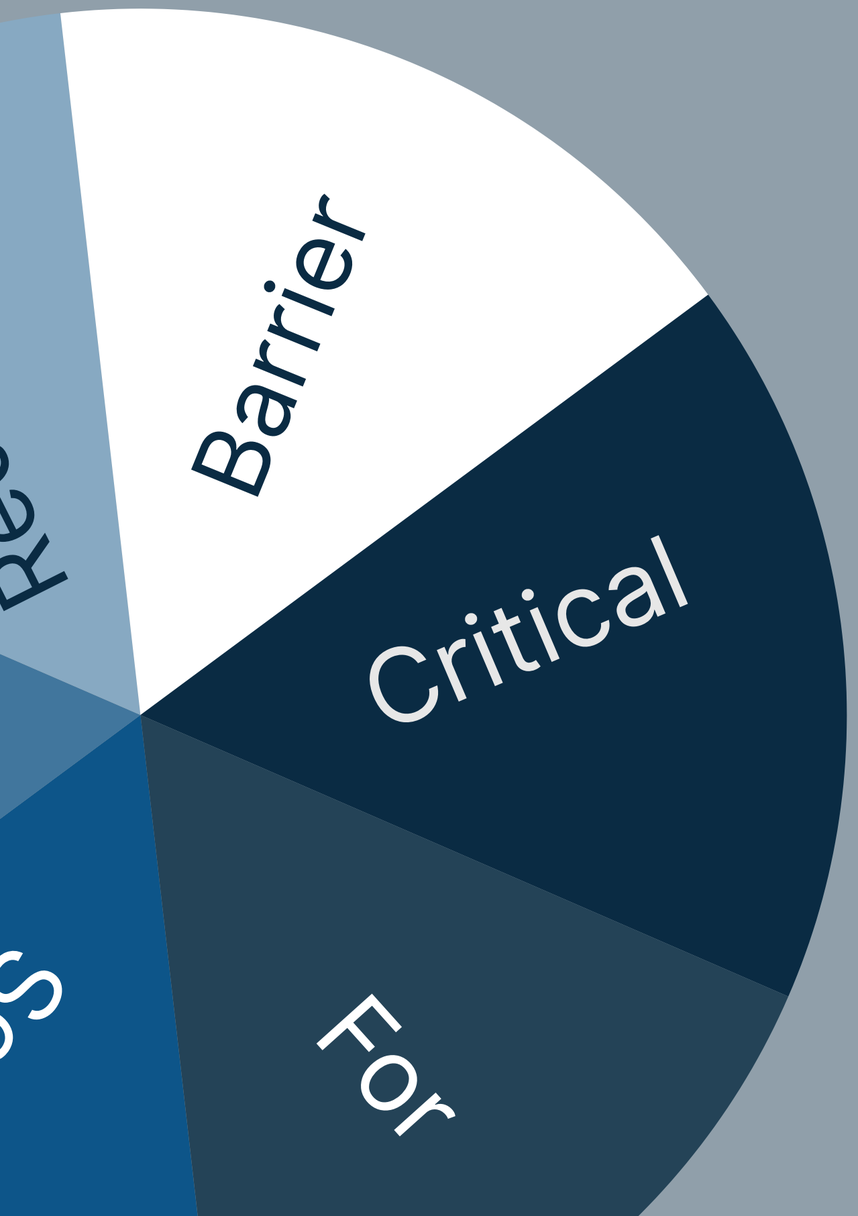
Clasulas y sincronización




```c
#pragma omp parallel for
for(int i=0; i<8; i++)
    printf("i=%d ejecutado por "
           "hilo %d\n",
           i, omp_get_thread_num());
```

```
s/Programas$ gcc -fopenmp for.c -o test.out
s/Programas$ ./test.out
```

```
i=2 ejecutado por hilo 1
i=3 ejecutado por hilo 1
i=4 ejecutado por hilo 2
i=5 ejecutado por hilo 2
i=6 ejecutado por hilo 3
i=7 ejecutado por hilo 3
i=0 ejecutado por hilo 0
i=1 ejecutado por hilo 0
```

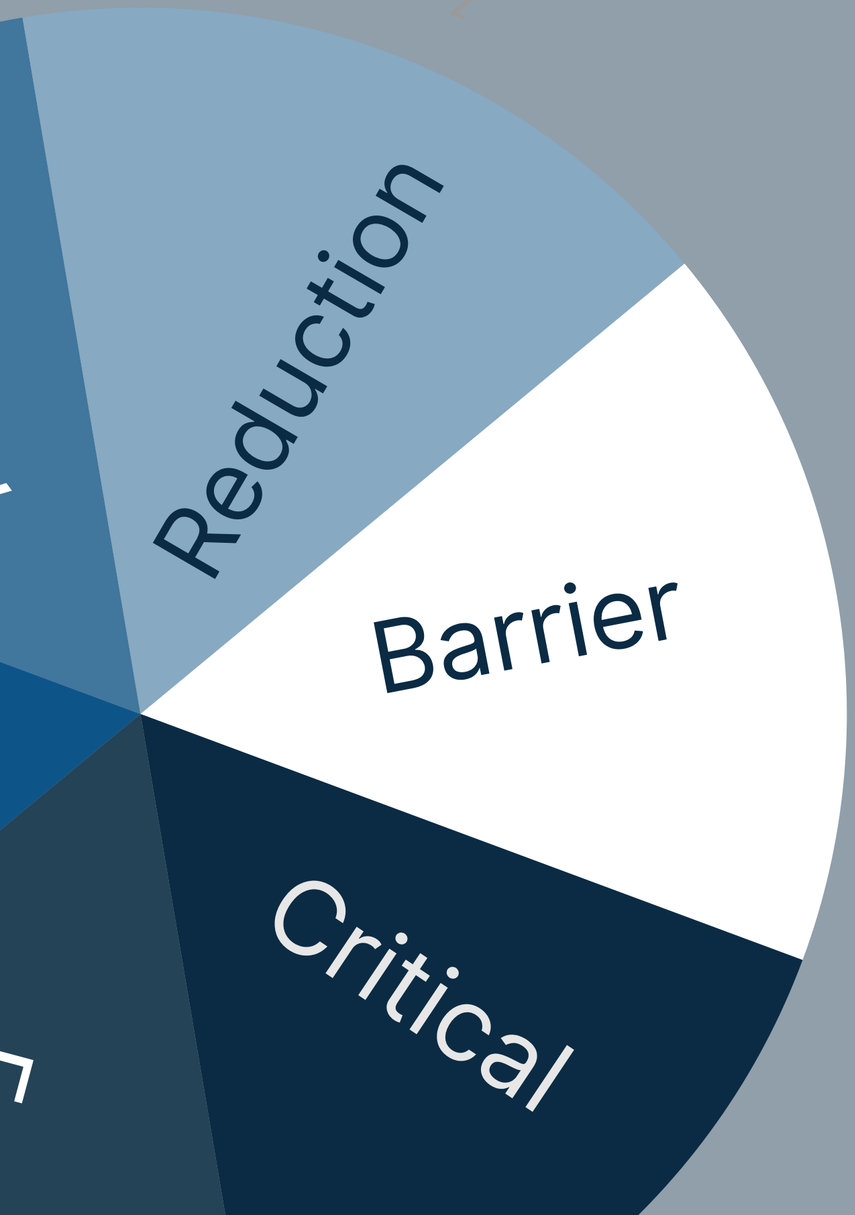


Clasulas y sincronización

```c
int contador = 0;
omp_set_num_threads(4); // Se puede fijar el número de hilos
#pragma omp parallel for
    for (int i = 0; i < 20; i++) {
        #pragma omp critical
        {
          contador++;
          printf("Hilo %d incrementó el contador a %d (iteración %d)\n",
          omp_get_thread_num(), contador, i);
        }
    }
printf("Valor final del contador = %d\n", contador);
```

```
/Programas$ gcc -fopenmp critical.c -o test.out
/Programas$ ./test.out
Hilo 3 incrementó el contador a 1 (iteración 8)
Hilo 3 incrementó el contador a 2 (iteración 9)
Hilo 1 incrementó el contador a 3 (iteración 3)
Hilo 1 incrementó el contador a 4 (iteración 4)
Hilo 1 incrementó el contador a 5 (iteración 5)
Hilo 2 incrementó el contador a 6 (iteración 6)
Hilo 2 incrementó el contador a 7 (iteración 7)
Hilo 0 incrementó el contador a 8 (iteración 0)
Hilo 0 incrementó el contador a 9 (iteración 1)
Hilo 0 incrementó el contador a 10 (iteración 2)
Valor final del contador = 10
```

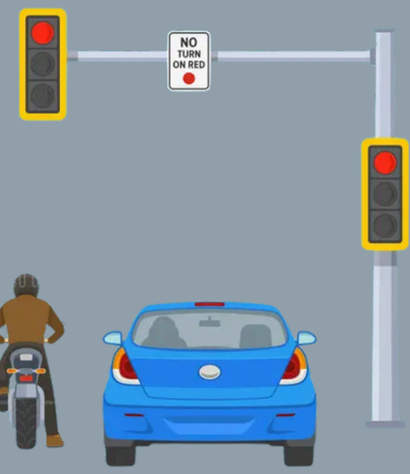
Clasulas y sincronización



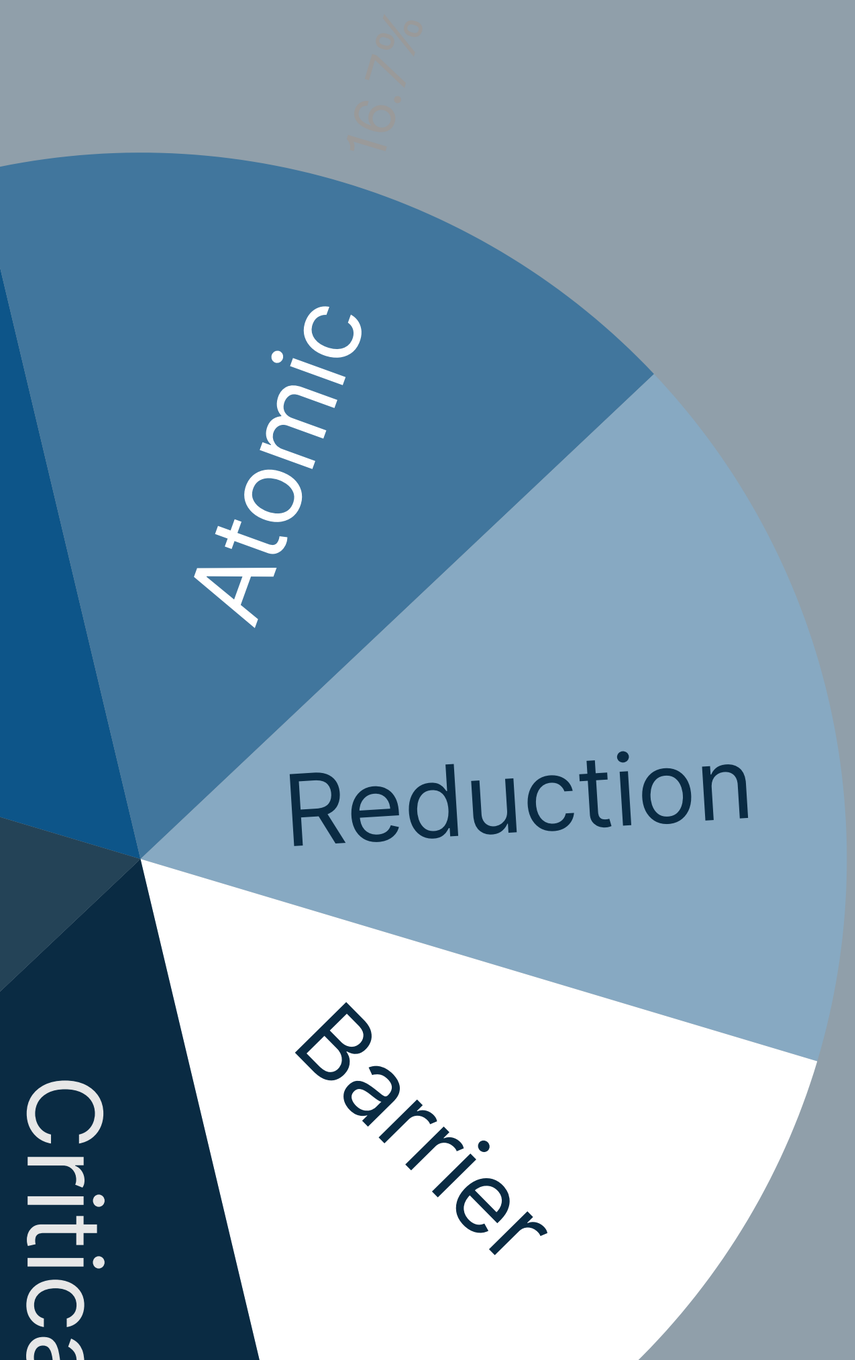

```c
#pragma omp parallel num_threads(4)
{
  printf("Primera parte\\n");
  #pragma omp barrier
  printf("Segunda parte\\n");
}
```

```
Primera parte
Primera parte
Primera parte
Primera parte
Segunda parte
Segunda parte
Segunda parte
Segunda parte
```

```
piaAndres/Programas$ gcc -fopenmp barrier.c -o test.out
piaAndres/Programas$ ./test.out
```



Clasulas y sincronización

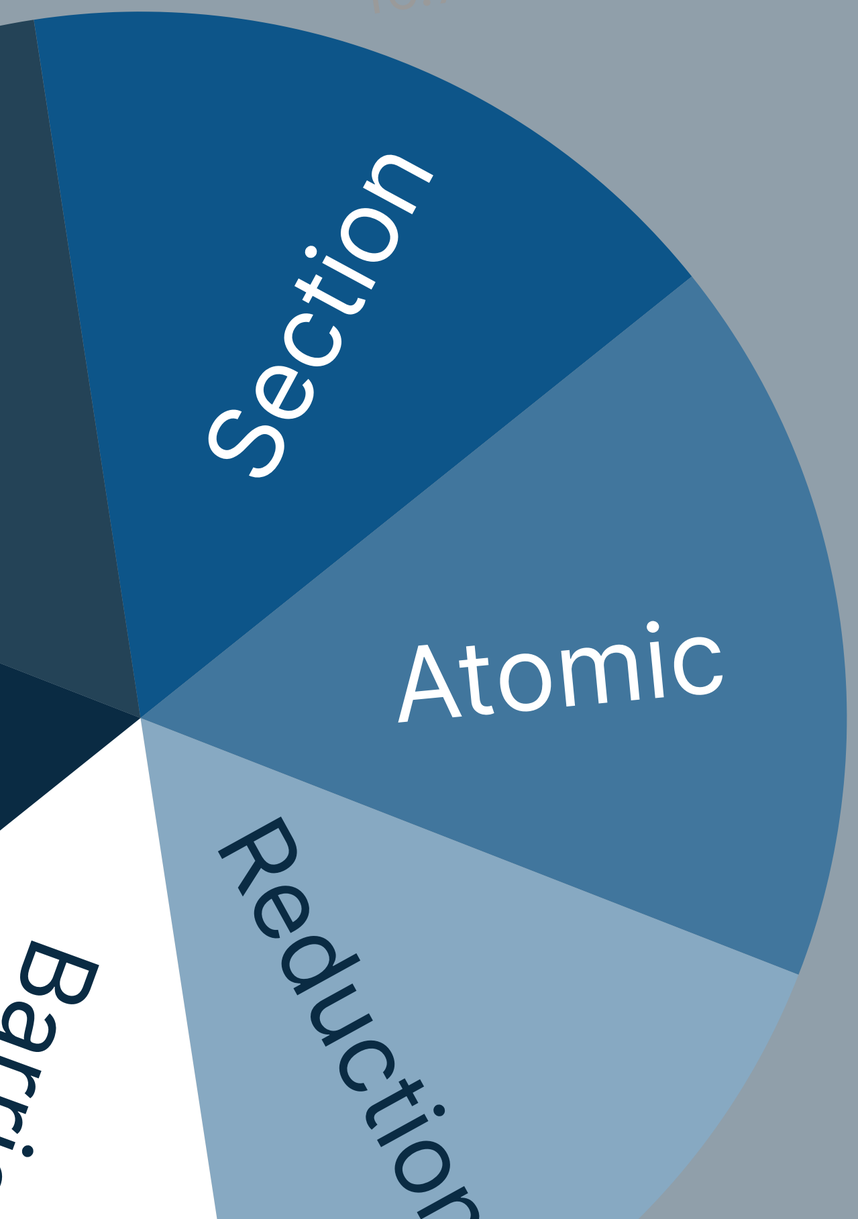



```c
int suma=0;
#pragma omp parallel for reduction(+:suma)
for (int i=0;i<100000;i++)
   suma++;
printf("Suma total: %d\\n",suma);
```

```
/Programas$ gcc -fopenmp reduction.c -o test.out
/Programas$ ./test.out
```

```
Suma total: 100000
```

Clasulas y sincronización




```c
int contador = 0;

#pragma omp parallel for
for (int i = 0; i < 1000; i++) {
  #pragma omp atomic
  contador++;
}
printf("Valor final: %d\n", contador);
```

```
s/Programas$ gcc -fopenmp atomic.c -o test.out
s/Programas$ ./test.out
```

```
Valor final: 1000
```

Paralelizado contra Secuencial

Comparativa: Secuencial vs Paralelo

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

int main() {
    long N = 100000000; // tamaño del vector
    double *vector = malloc(N * sizeof(double));

    if (vector == NULL) {
        printf("Error al asignar memoria\n");
        return 1;
    }

    // Inicializar el vector
    for (long i = 0; i < N; i++) {
        vector[i] = 1.0;
    }
```

```c
/* ================================
 *    SUMA SECUENCIAL
 * ================================ */
double t0 = omp_get_wtime();
double suma_secuencial = 0.0;

for (long i = 0; i < N; i++) {
    suma_secuencial += vector[i];
}

double t1 = omp_get_wtime();
double tiempo_secuencial = t1 - t0;
```

```c
/* ================================
 *    SUMA PARALELIZADA
 * ================================ */
int num_hilos = 4;
double suma_paralela = 0.0;

double t2 = omp_get_wtime();

#pragma omp parallel num_threads(num_hilos)
{
    double suma_local = 0.0;

    #pragma omp for
    for (long i = 0; i < N; i++) {
        suma_local += vector[i];
    }

    #pragma omp atomic
    suma_paralela += suma_local;
}

double t3 = omp_get_wtime();
double tiempo_paralelo = t3 - t2;
```

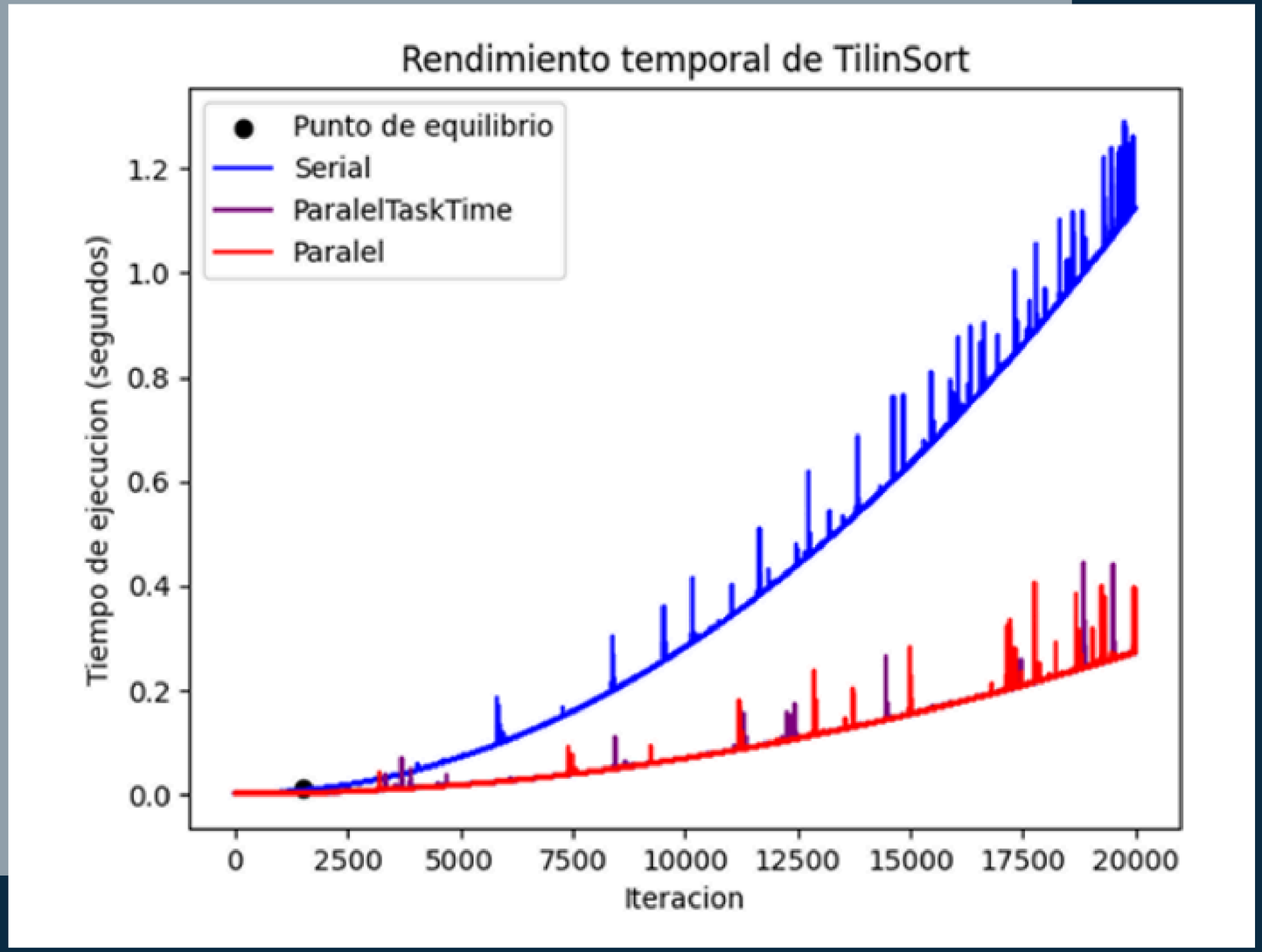
```c
    /* ================================
     *    RESULTADOS
     * ================================ */
    printf("Suma secuencial: %f\n", suma_secuencial);
    printf("Tiempo secuencial: %f segundos\n\n", tiempo_secuencial);

    printf("Suma paralela:   %f\n", suma_paralela);
    printf("Tiempo paralelo: %f segundos\n", tiempo_paralelo);

    free(vector);
    return 0;
}
```

Resultados

```
andrestg@192:~/Respaldo_Repositorios/programasC$ ls
vs.c
andrestg@192:~/Respaldo_Repositorios/programasC$ gcc -fopenmp vs.c -o comparativa
andrestg@192:~/Respaldo_Repositorios/programasC$ ./comparativa
Suma secuencial: 100000000.000000
Tiempo secuencial: 0.415696 segundos

Suma paralela:   100000000.000000
Tiempo paralelo: 0.142633 segundos
andrestg@192:~/Respaldo_Repositorios/programasC$
```

Consideraciones

La paralelización no siempre conviene: en tareas pequeñas es más lenta y costosa. Es clave conocer el punto donde paralelizar realmente mejora el rendimiento.



Referencias

1. Informatec Digital. (2025). Openmp: Qué es, cómo funciona y todo lo que necesitas saber. Descargado de https://informatecdigital.com/que-es-openmp/
2. Linux Kernel Development Community. (2025). Gestor de tareas cfs: Documentación oficial. Descargado de https://www.kernel.org/doc/html/latest/translations/sp_SP/scheduler/sched-design-CFS.html
3. LinuxVox. (2025). task_struct en el kernel de linux. Descargado de https://linuxvox.com/blog/task_struct-linux/
4. Microsoft. (2025). Openmp directives. Descargado de https://learn.microsoft.com/es-es/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170 (Consultado el 11 de noviembre de 2025)
5. Swahn, H. (2016). Pthreads and openmp: A study on the relationship between posix threads and openmp. Descargado de https://www.diva-portal.org/smash/get/diva2%3A944063/FULLTEXT02
6. Tapia, A. (2023). Proyecto02- versiones paralelas de un algoritmo serial. (Curso Estructura de Datos II [Profesor: Solano Galvez Jorge])
7. Whileint. (2021). ¿qué es la llamada al sistema de clonación en linux? Descargado de https://whileint.com/tech/pablo/que-es-la-llamada-al-sistema-de-clonacion-en-linux/
8. Wolf, G. (2025a). Planificación de procesos (sistemas operativos 2025.10.21). Video en YouTube. Descargado de https://www.youtube.com/watch?v=NUCwBGS0aj8
9. Wolf, G. (2025b, sep). Sistemas operativos 2025.09.04: Administración de procesos. Video en YouTube. Descargado de https://youtu.be/34QOTfafCI4 (Consultado el 18 de noviembre de 2025)