# Laboratorio 05

***Competencias para desarrollar***

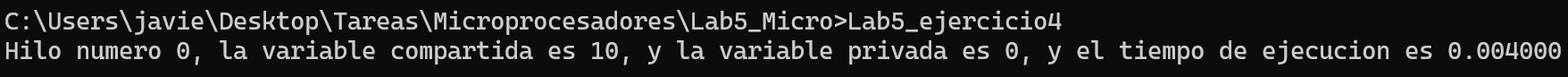
Distribuir la carga de trabajo entre hilos utilizando programación en C y OpenMP.

***Instrucciones***

Esta actividad se realizará individualmente. Al finalizar los períodos de laboratorio o clase, deberá entregar este archivo en formato PDF y los archivos .c en la actividad correspondiente en Canvas.

1. **(18 pts.)** Explica con tus propias palabras los siguientes términos:
2. Private: Es una forma de declarar variables que hace que cada hilo tenga su propia copia de la variable y que los cambios que un hilo le haga a esa variable no afectan a la copia de los otros hilos.
3. Shared: Este tipo de variables son compartidas entre todos los hilos, esto quiere decir que todos pueden verlas y modificarlas. Hay que ser cuidadosos al usarlas porque si varios hilos las modifican al mismo tiempo puede haber problemas.
4. Firstprivate: Son como las variables private, pero la diferencia es que inician con el valor que tenían antes de la paralelización. Después de esto cada hilo tiene su copia de la variable.
5. Barrier: Es una barrera que pone un tope en la ejecución hasta que todos los hilos lleguen a ese punto. Sirve para asegurar que todos los hilos terminaron su trabajo antes de seguir con la ejecución.
6. Critical: Se usa en un bloque de código para que solo un hilo lo ejecute a la vez. Sirve para evitar que varios hilos accedan a la misma parte del código.
7. Atomic: Se utiliza para operaciones muy simples, el estándar es usarlo para una sola línea de código. Asegura que una operación no sea interrumpida por otros hilos.
8. **(12 pts.)** Escribe un programa en C que calcule la suma de los primeros N números naturales utilizando un ciclo ***for* paralelo**. Utiliza la cláusula **reduction con +** para acumular la suma en una variable compartida.
9. Define N como una constante grande, por ejemplo, N = 1000000.
10. Usa omp\_get\_wtime() para medir los tiempos de ejecución.
11. **(15 pts.)** Escribe un programa en C que ejecute tres funciones diferentes en paralelo usando la **directiva #pragma omp sections**. Cada sección debe ejecutar una función distinta, por ejemplo, una que calcule el factorial de un número, otra que genere la serie de Fibonacci, y otra que encuentre el máximo en un arreglo, operaciones matemáticas no simples. Asegúrate de que cada función sea independiente y no tenga dependencias con las otras.
12. **(15 pts.)** Escribe un programa en C que tenga un ciclo for donde se modifiquen dos variables de manera paralela usando #pragma omp parallel for.
    1. Usa la cláusula shared para gestionar el acceso a la variable1 dentro del ciclo.
    2. Usa la cláusula private para gestionar el acceso a la variable2 dentro del ciclo.
    3. Prueba con ambas cláusulas y explica las diferencias observadas en los resultados.

Diferencias:



Al utilizar “shared” es posible que se generen race conditions, por lo que el resultado final puede ser 10 o un número menor por posibles problemas de unificación. Al utilizar private se evitan las race conditions, por lo que evitamos este problema, pero al cada hilo tener su copia de la variable, puede pasar que al salir del ciclo el valor de la variable se reinicie al valor inicial.

1. **(30 pts.)** Analiza el código en el programa Ejercicio\_5A.c, que contiene un programa secuencial. Indica cuántas veces aparece un valor key en el vector a. Escribe una versión paralela en OpenMP utilizando una descomposición de tareas iterativa, en la cual se generen tantas tareas como hilos.
2. **REFLEXIÓN DE LABORATORIO: se habilitará en una actividad independiente.**