Casa abierta al tiempo

Computo concurrente

Practica 1

Suma concurrente

- Alejandro Chavez Flores
  2203024955

- Rodolfo Andre Ortega Garcia
  2203066275

Condición de Competencia.

La condición de competencia es un comportamiento del Software en el cual la salida depende de un orden de ejecución de eventos que no se encuentran bajo control y que puede provocar resultados incorrectos. Por lo tanto:

- Se crean fallos cuando el orden de ejecución no es el esperado.
- El nombre viene de que hay dos procesos o en este caso hilos que compiten para acceder a un recurso compartido.

Sección Crítica.

Se trata de la parte del código de un programa de ordenador en la que se accede a un recurso compartido que no debe ser accedido por más de un proceso o hilo en ejecución.

Nota: Por lo que debemos de estar conscientes en qué si se tiene una variable compartida y se quiere acceder a ella, se tiene que hacer uso de los candados para poder lograr una sincronización adecuada en los procesos. Ya que, si no se sincronizan los procesos, siempre estarán compitiendo por ver quien será el que acceda a la variable global.

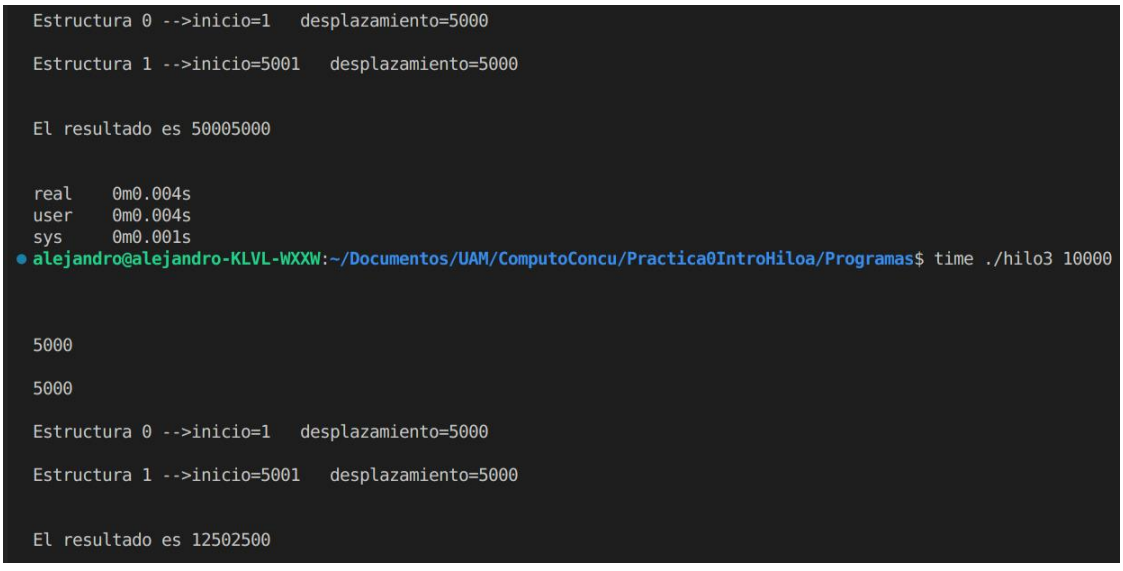

Imagen 1. Se muestran dos ejecuciones del programa, pero sin candados. Vemos como cambia el valor del resultado por la condición de competencia. Por lo tanto, hay una sección crítica.

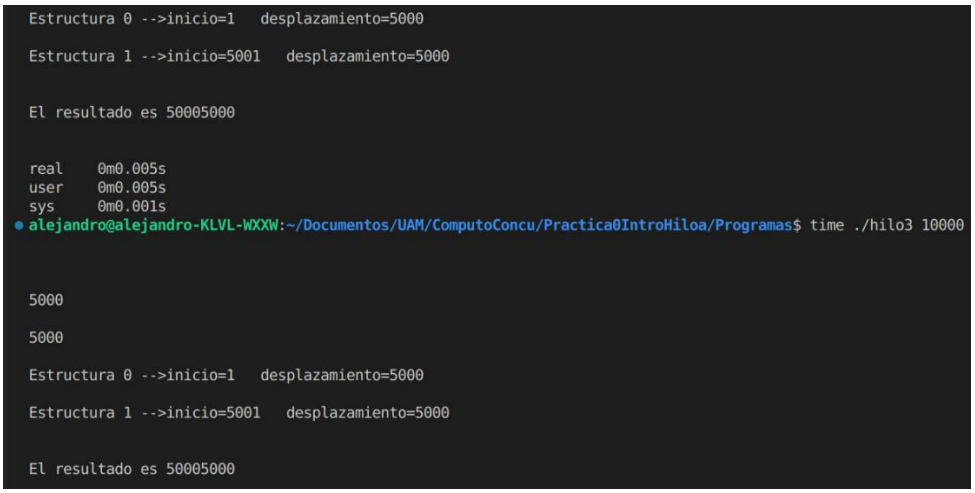

Imagen 2. Se muestran dos ejecuciones, pero ahora con candados. Se puede observar que el programa se ejecuta de la manera esperada porque nuestros hilos están sincronizados.

Para poder hacer que nuestro programa funcionará al momento de usar nuestra variable global sin tener el riesgo de que tuviéramos la sección crítica son los siguientes pasos:

1. Crear una variable global de tipo **pthread_mutex_t**. La cuál será nuestro candado.
2. En nuestra función principal main inicializar nuestro candado con la función: **pthread_mutex_init(&nombre_candado,NULL).**
3. En la parte del programa donde se hará uso de la variable compartida se utilizará la **función lock(&nombre_candado)** esto con el fin de proteger al momento que un proceso entra a modificar la variable compartida, cuando se acabe de utilizar la variable se utiliza la función **unlock(&nombre_candado)** esto para poder abrir el candado que se utilizo previamente.

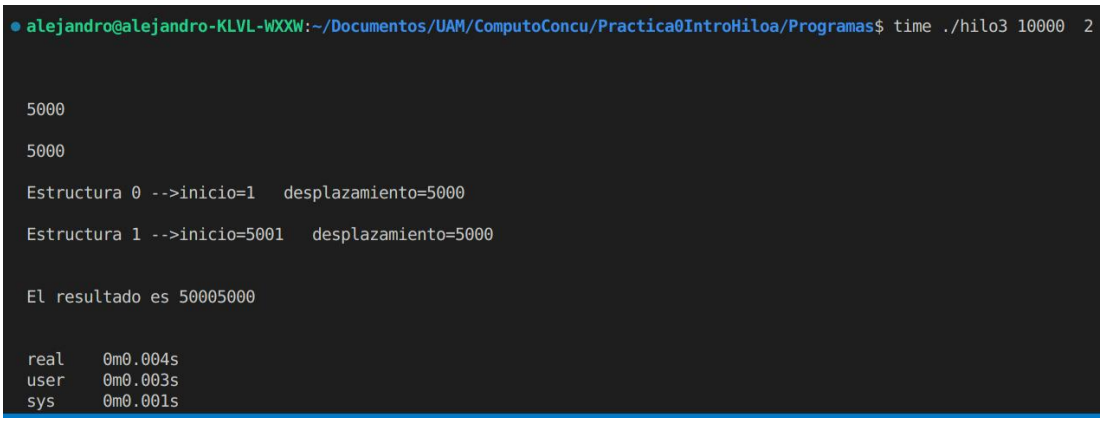

Imagen 3. Se muestra la ejecución considerando un arreglo de 10,000 elementos y 2 hilos para ejecutar la suma.

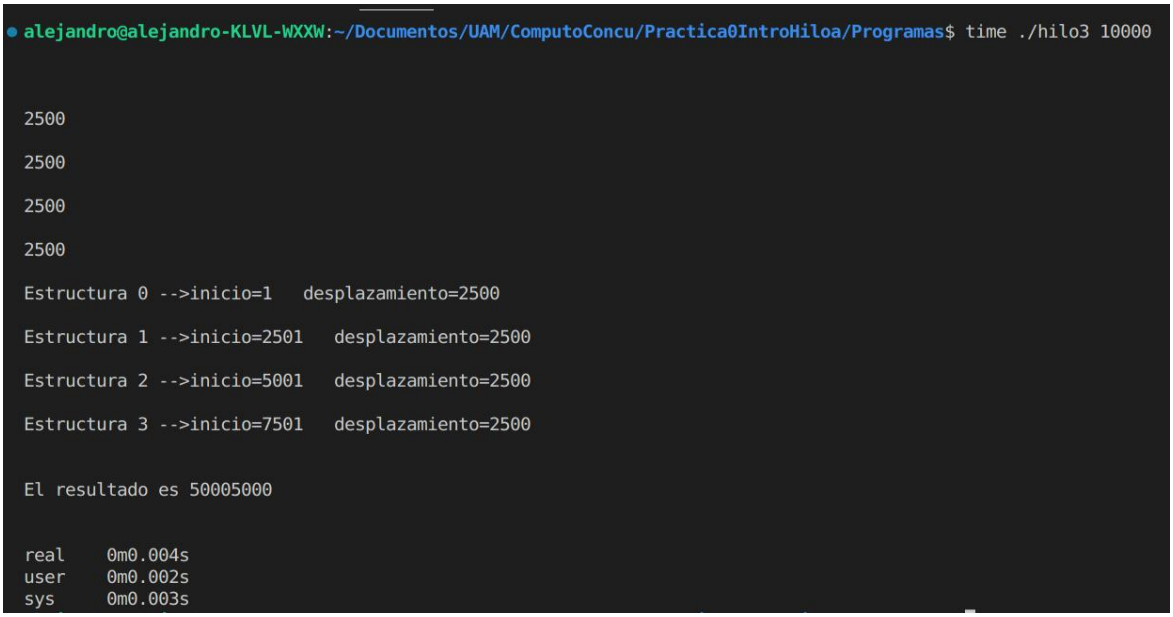

Imagen 4. Se muestra la ejecución considerando un arreglo de 10,000 elementos y 4 hilos para ejecutar la suma.

Conclusión.

Vemos que las dos ejecuciones toman tiempos similares ya que la suma es un proceso muy rápido de procesar y de hecho se puede ejecutar más rápido en un solo hilo que compartiendo el proceso de sumas con n hilos.