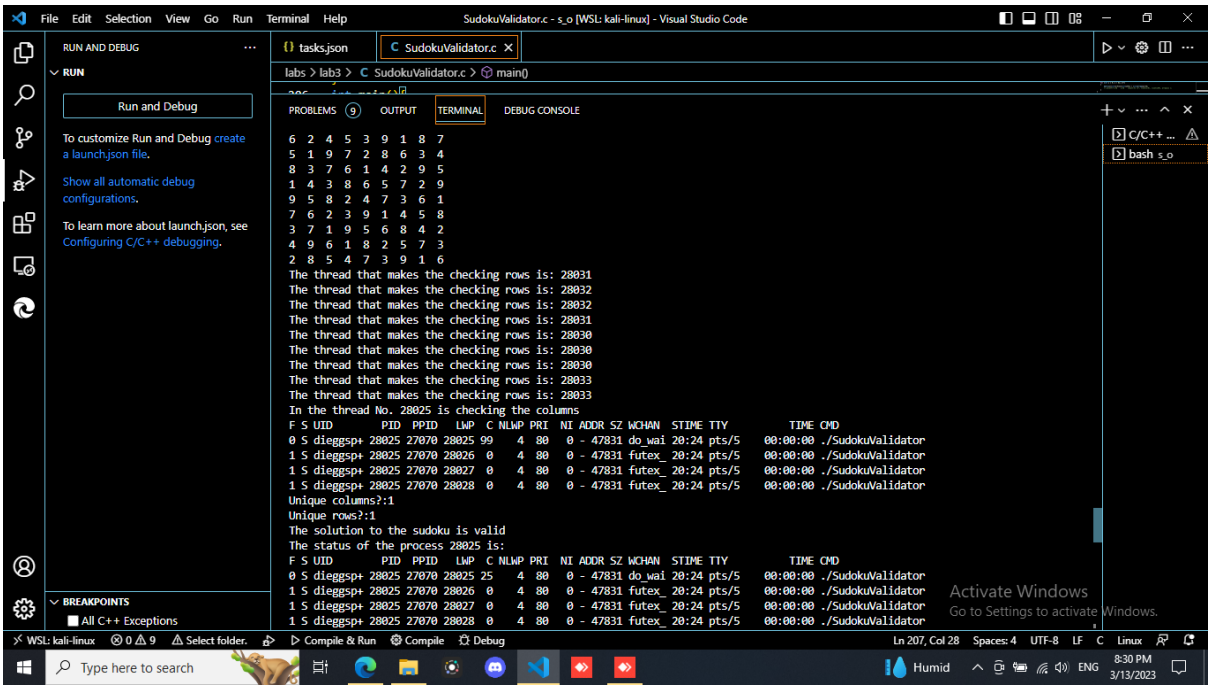
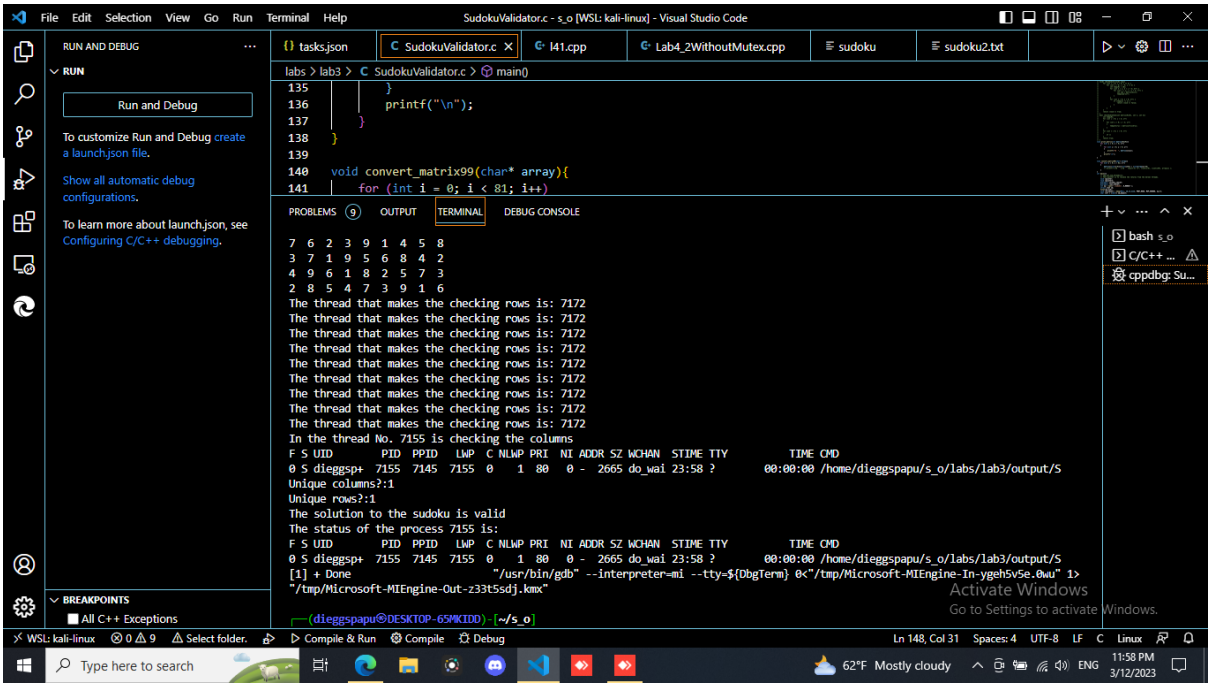
Diego Andrés Alonzo Medinilla 20172

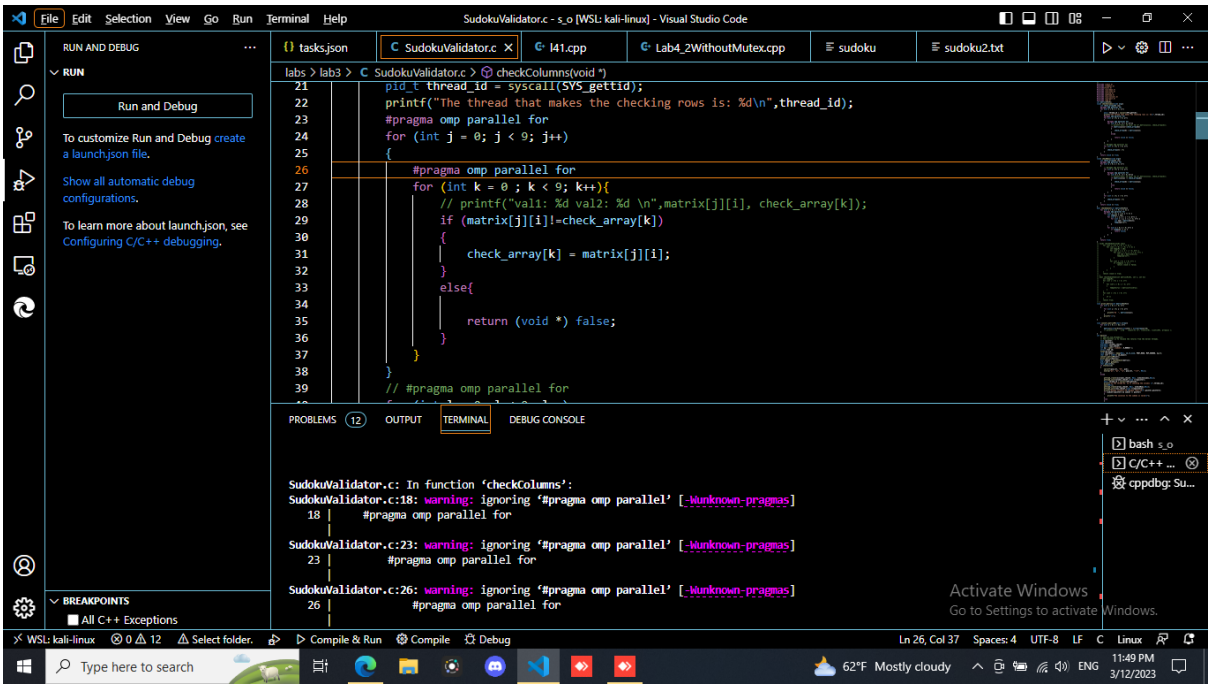
Lab3 SO



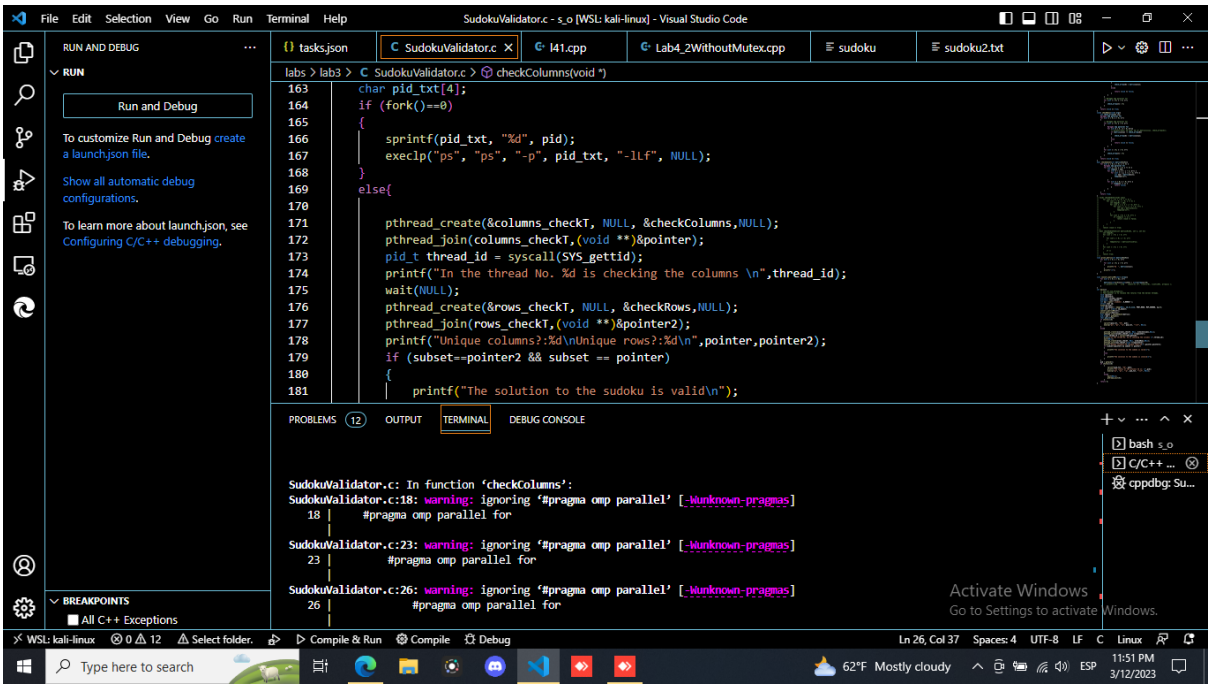


1. ¿Qué es una race condition y porqué hay que evitarlas? Una race condition es una condición donde dos o más threads pueden acceder a cierta memoria compartida generando que se modifique dicha memoria de manera que no sea lo esperable.
2. ¿Cuál es la relación, en Linux, entre pthreads y clone()? ¿Hay diferencia al crear threads con uno o con otro? ¿Qué es más recomendable? Un pthread o posix thread es un lightweight process o un proceso de bajo consumo, así mismo, estos son creados a partir de funciones de pthreads, sin embargo, en el background realmente se ejecuta una llamada a sistema de clone que sirve para crear/clonar procesos y poder indicarle a un proceso que recursos compartirá, así como de datos y demás. Realmente no existe una diferencia al crear uno u otro.

3. ¿Dónde, en su programa, hay paralelización de tareas, y dónde de datos? En el caso de mi programa la paralelización de tareas ocurre en:

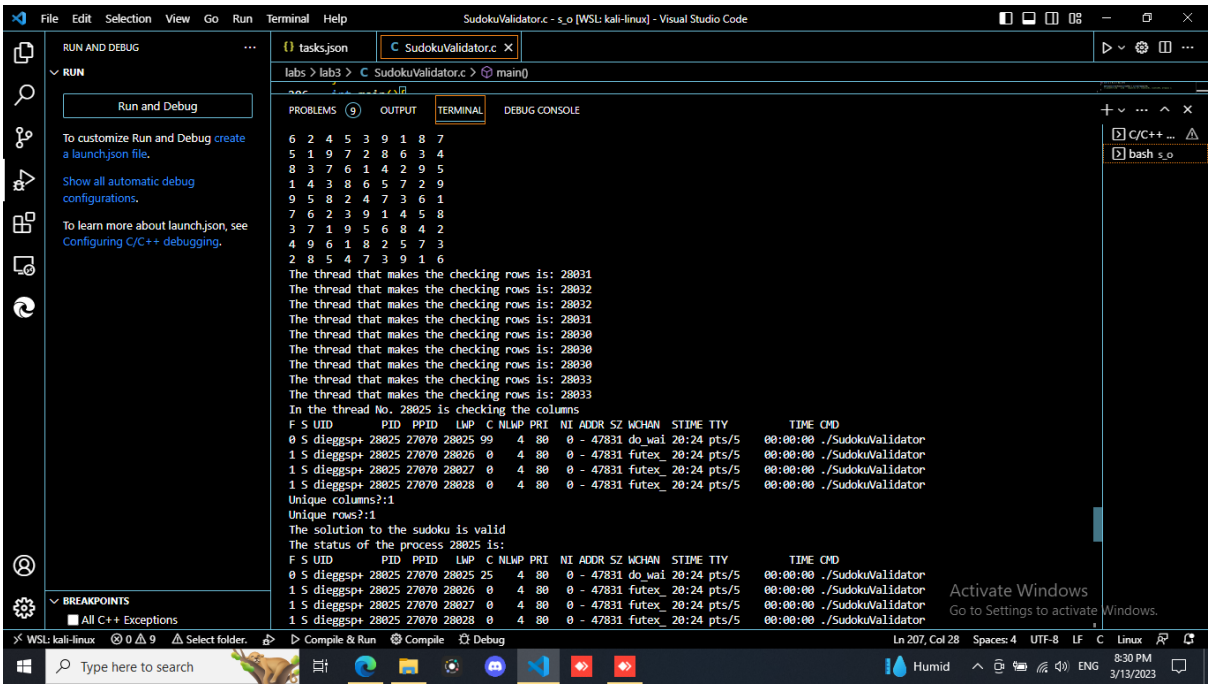


Dado que se asigna la tarea de verificar que estén todos los números del uno al nueve en la columna.
Y se realiza la paralelización de datos al momento de realizar el fork debido a que se divide y se ejecutan diferentes tareas para datos distintos.

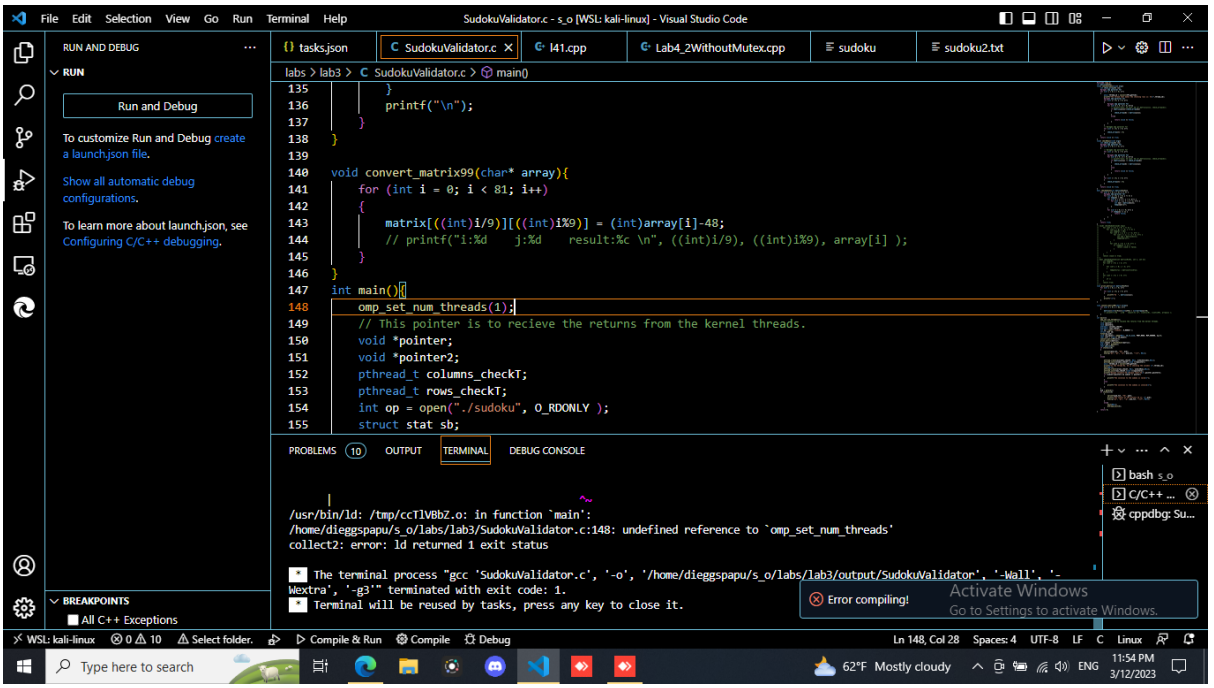


4. Al agregar los #pragmas a los ciclos for, ¿cuántos LWP's hay abiertos antes de terminar el main()y cuántos durante la revisión de columnas? ¿Cuántos user threads deben haber abierto en cada caso, entonces? Hint: recuerde el modelo de multithreading que usan Linux y Windows.
Tal como se observa la cantidad de lightweight process abiertos antes de terminar el

main es de 4 y de revisión de las columnas también es de 4. Así mismo, los user threads que debieron abrir en cada caso debió de ser de 4 dado que el modelo que implementa el modelo de Linux y Windows es de 1 a 1.

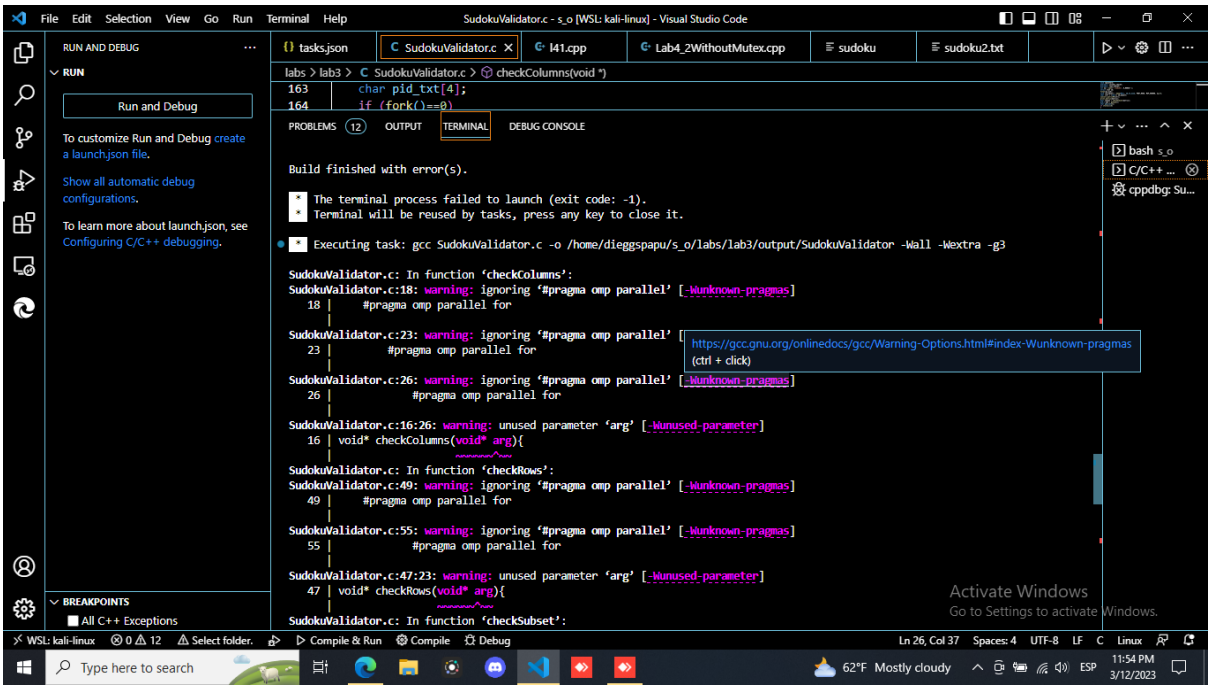


Se abrieron 4 threads de LWP
Solo quería destacar que estos errores me aparecían.

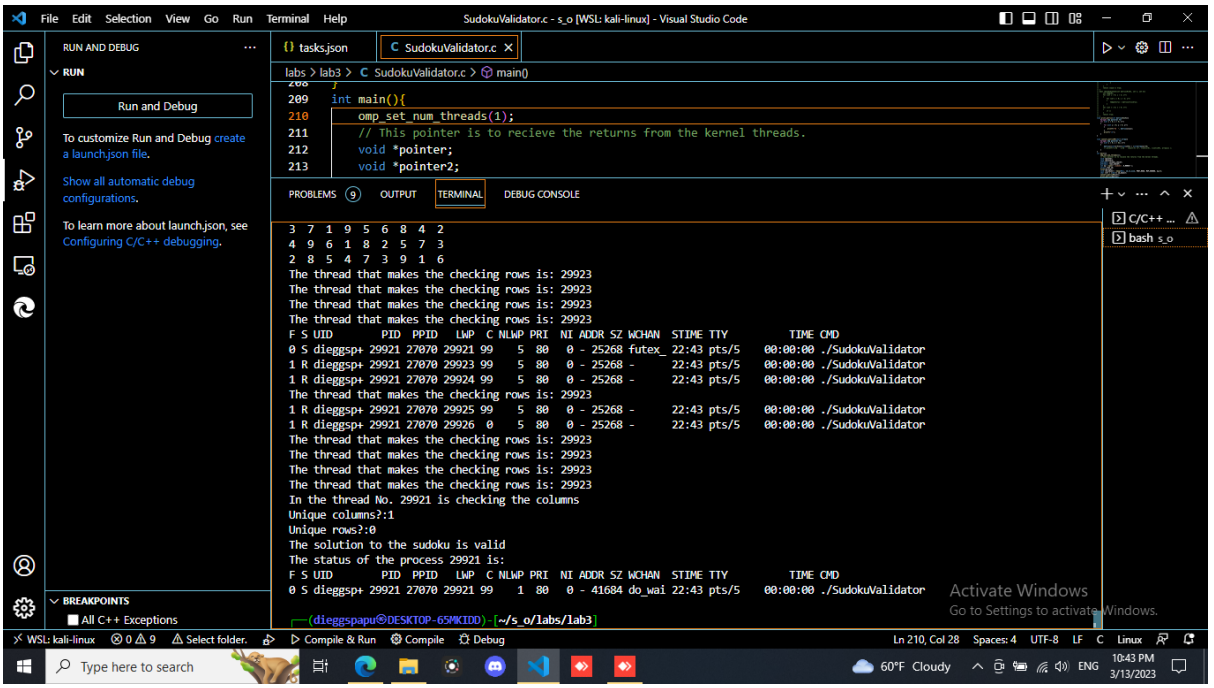


y en el caso de la paralelización por for's no servía por estos warnings donde en sí el

compilador no reconocía el pragma

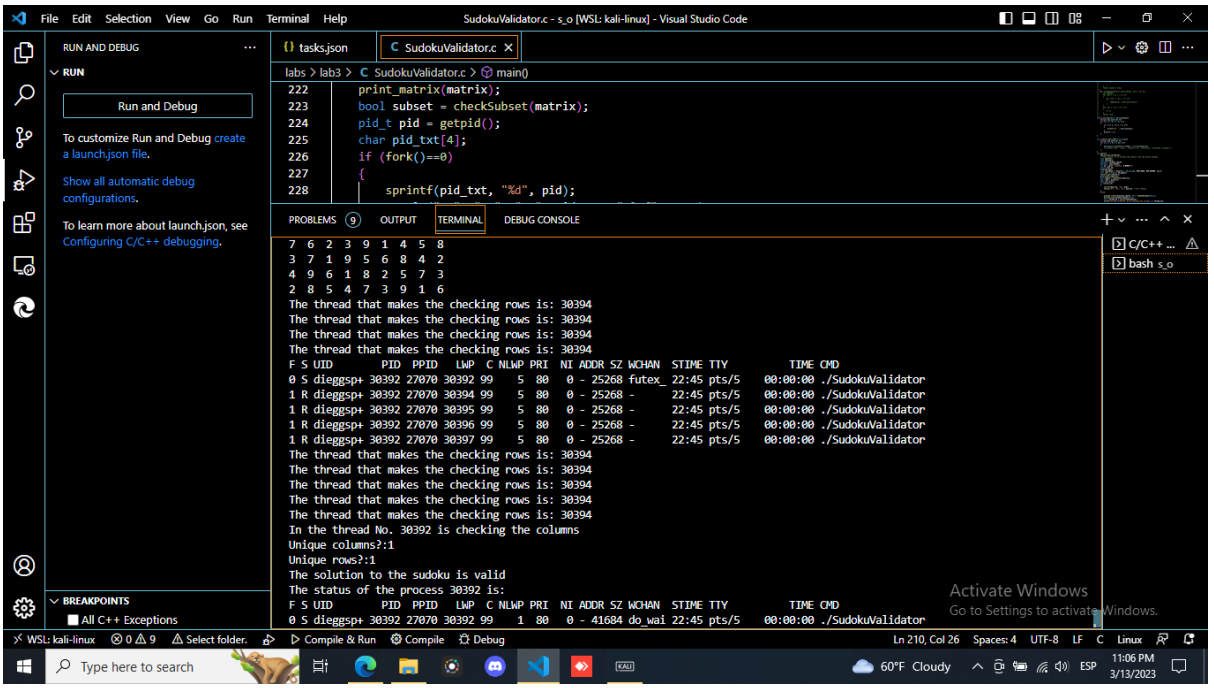


5. Al limitar el número de threads en main() a uno, ¿cuántos LWP's hay abiertos durante la revisión de columnas? Compare esto con el número de LWP's abiertos antes de limitar el número de threads en main(). ¿Cuántos threads (en general) crea OpenMP por defecto?



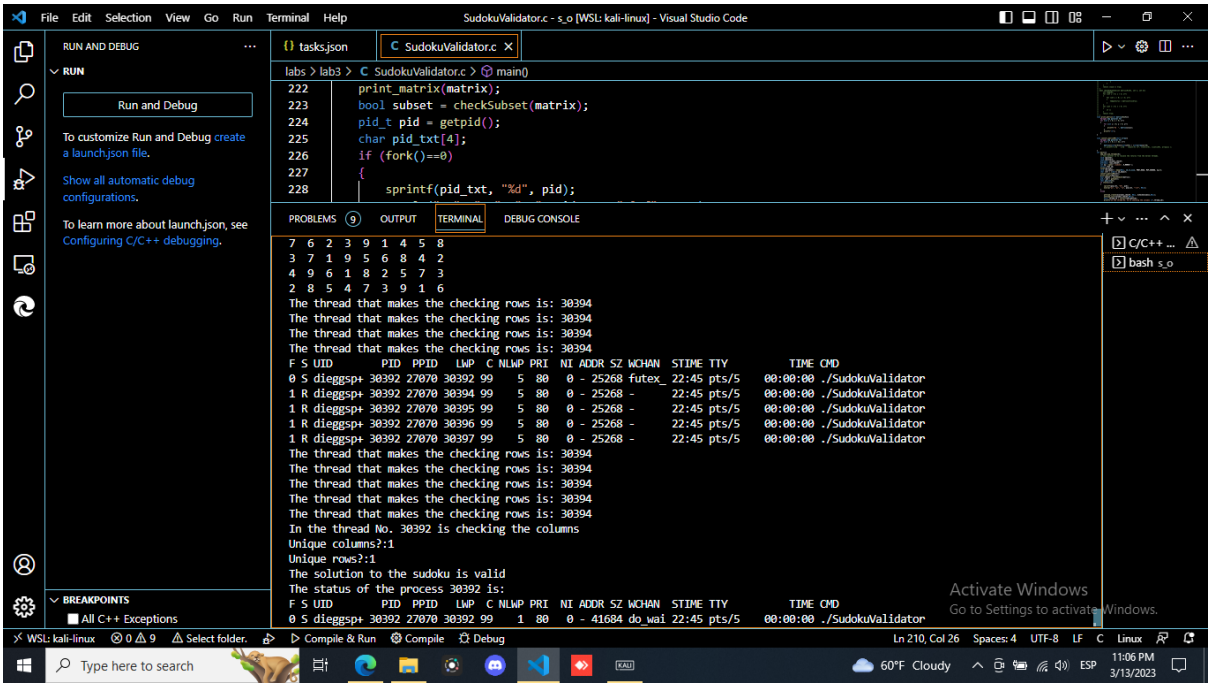
La cantidad de lightweights process que crea es de 5 por durante la revisión de las columnas, mientras que en el caso del número de threads al final es de solamente uno. De manera que contrasta ya que al momento de limitar la cantidad de threads se genera un modelo de multitasking similar al modelo de uno ya que crea 5 threads pero luego al pasar a la ejecución del kernel se asigna a solo uno, generando 1 único thread. Así mismo, al momento de crear los hilos crea 4 por defecto.

6. Observe cuáles LWP's están abiertos durante la revisión de columnas según ps. ¿Qué significa la primera columna de resultados de este comando? ¿Cuál es el LWP que está inactivo y por qué está inactivo? Hint: consulte las páginas del manual sobre ps.
   La primera columna significa la flag del thread, en el caso de la flag que indica 1 significa que se realizó un fork pero no se realizó un exec. Así mismo, el LWP que está inactivo es aquel que dice FUTEX debajo de wchan, este indica que está bloqueado por una llamada a futex en el kernel, así mismo, se encuentra inactivo o en sleep porque es el parent id esperando a que termine de ejecutarse el execlp por parte del fork.

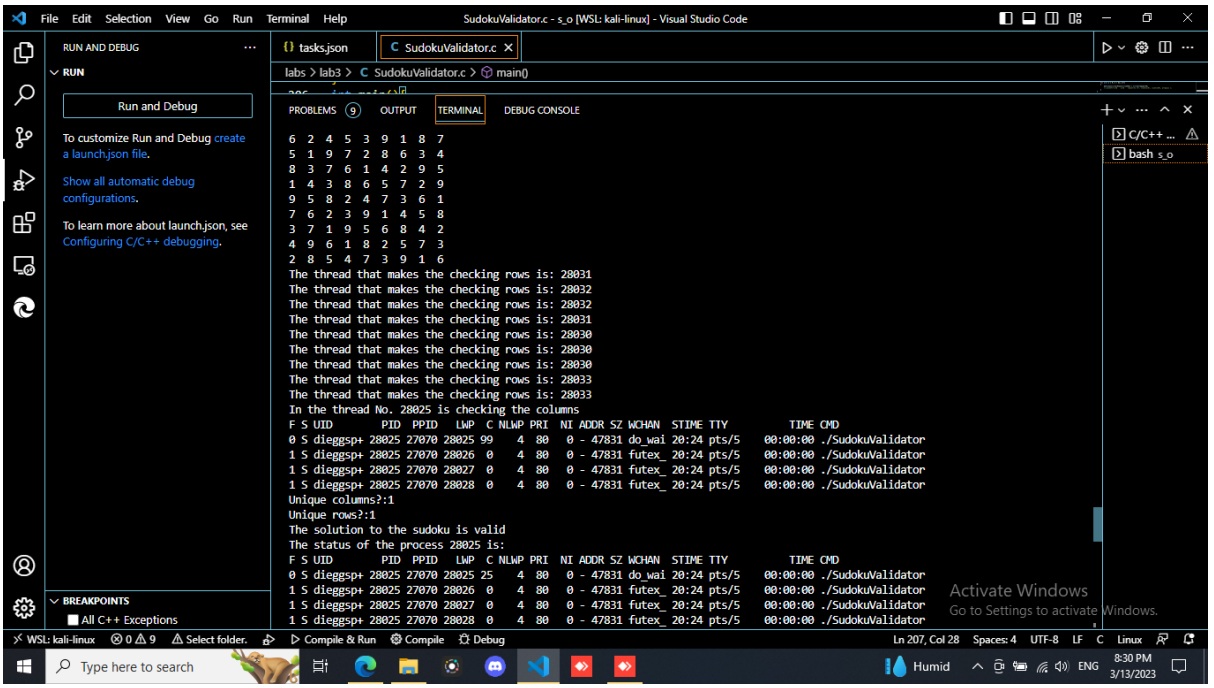


7. Compare los resultados de ps en la pregunta anterior con los que son desplegados por la función de revisión de columnas per se. ¿Qué es un thread team en OpenMP y cuál es el master thread en este caso? ¿Por qué parece haber un thread "corriendo", pero que no está haciendo nada? ¿Qué significa el término busy-wait? ¿Cómo maneja OpenMP su thread pool?
   Resultados dados por el ps usando el omp_set_num_threads(1):

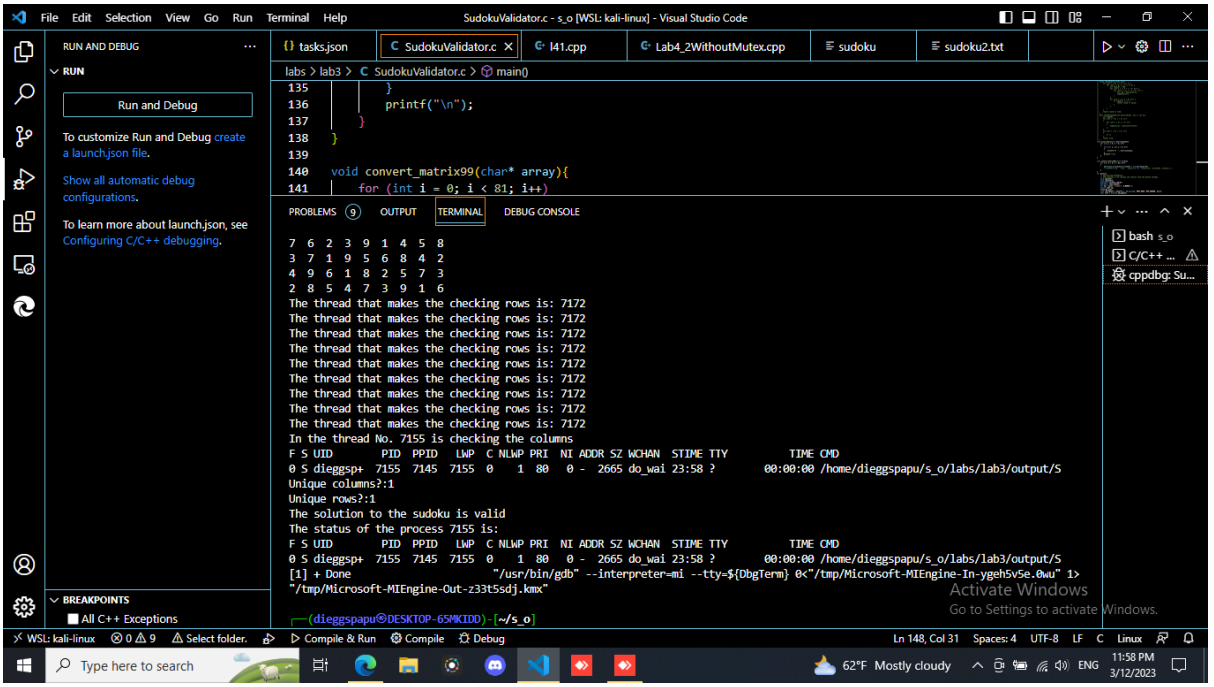
Diego Andrés Alonzo Medinilla 20172



Resultados dados con la paralelización:

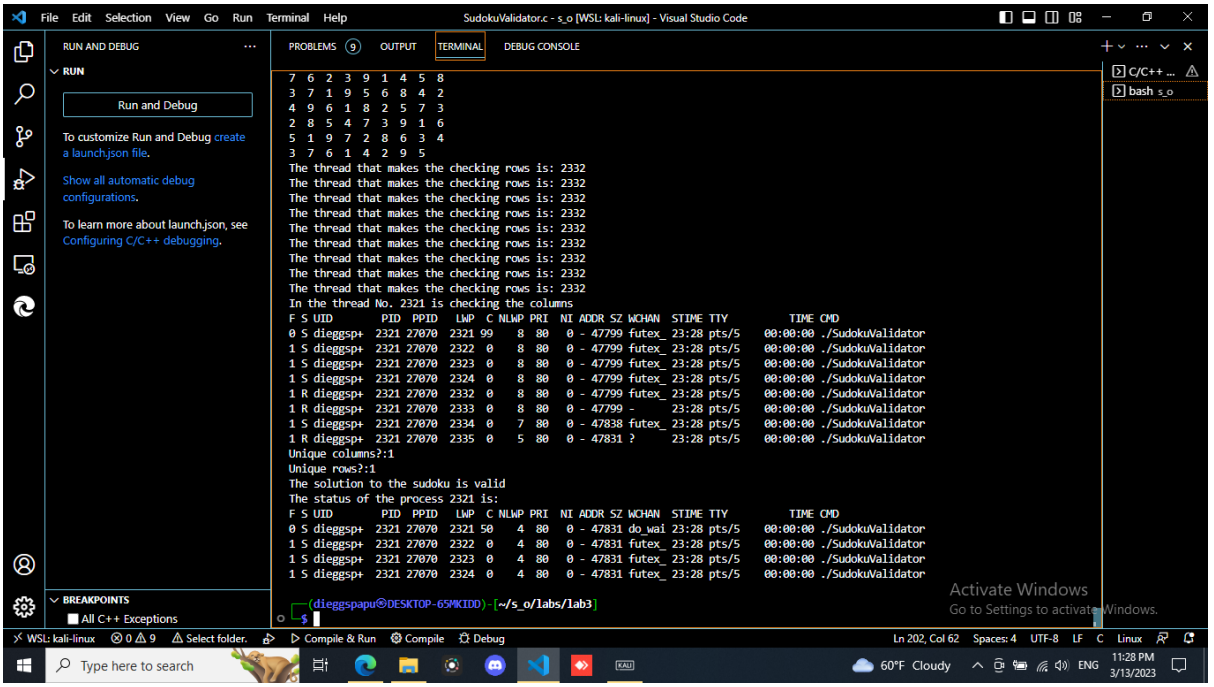


Resultados dados per se sin la paralelización:



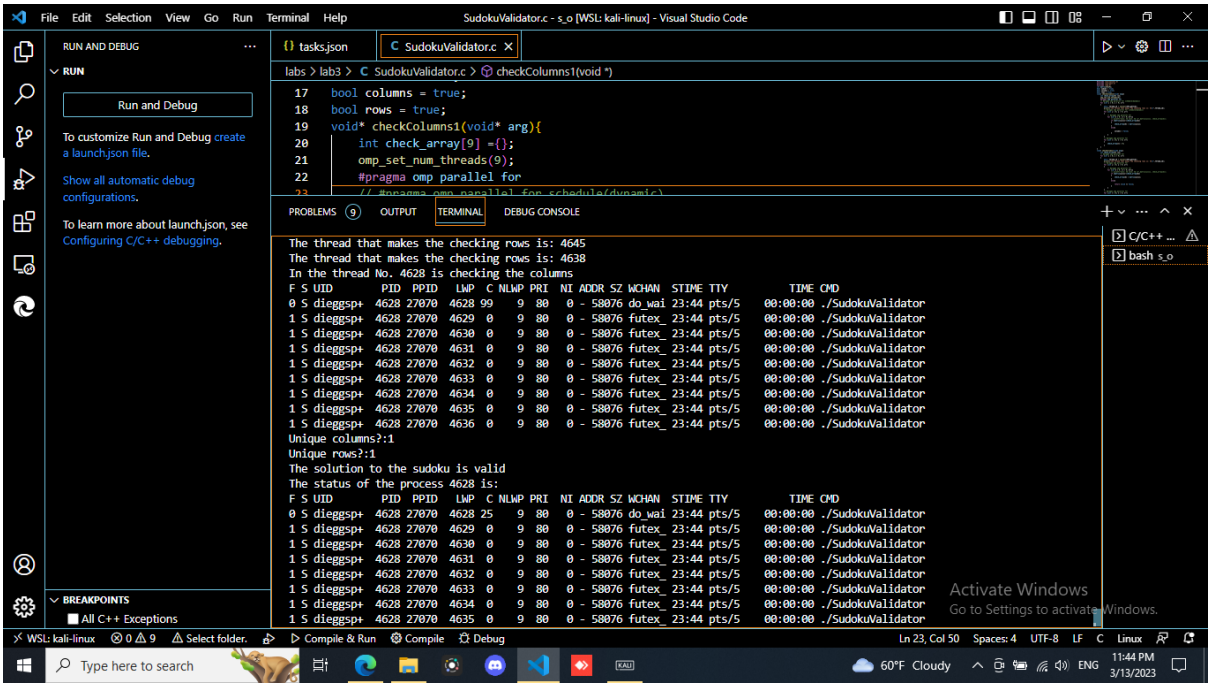
Se debe de notar que al momento de utilizar el per se se realiza un wait para esperar una cola, así mismo, en el caso de utilizar el paralelizado hay un thread que realiza la espera (que sería el parent) y los demás realizan una llamada a futex, y en el caso del thread único solamente 1 tiene un futex. Un thread team es cuando un thread entra a una región paralelizable de manera que el thread que entra se forkea y se genera un thread team, es similar a un fork-join de threading implícito. En el caso del thread que está corriendo, sin embargo, no está haciendo nada es debido a que es el main thread que está esperando a los otros. Busy wait es es una técnica de sincronización de procesos en la que un proceso espera y constantemente chequea por una condición para proseguir con su ejecución.Open MP maneja su thread pool creando un thread por cada core que está en la computadora, de esta forma mantiene el modelo uno a uno de multithreading.

8. Luego de agregar por primera vez la cláusula schedule(dynamic) y ejecutar su programa repetidas veces, ¿cuál es el máximo número de threads trabajando según la función de revisión de columnas? Al comparar este número con la cantidad de LWP’s que se creaban antes de agregar schedule(), ¿qué deduce sobre la distribución de trabajo que OpenMP hace por defecto?



El número máximo de threads según la revisión de columnas es 8, así mismo, al comparar este número con la cantidad de antes de que se agregara el schedule puedo suponer que se optimizan los recursos de manera que puede optimizarse de mejor manera ya que cuando se acabe de utilizar el thread para una iteración en el for, el mismo thread se puede asignar a la siguiente iteración realizando 2 veces su trabajo.

9. Luego de agregar las llamadas omp_set_num_threads() a cada función donde se usa OpenMP y probar su programa, antes de agregar omp_set_nested(true), ¿hay más o menos concurrencia en su programa? ¿Es esto sinónimo de un mejor desempeño? Explique



Hay más concurrencia debido a que lo que hace la función es settear la cantidad de threads que se crearán, de manera que en este caso desde un principio se crea la

cantidad de threads que se utilizarán para cada paralelización, es por ello que en este caso son 9. Sin embargo, esto no es sinónimo de un mejor desempeño ya que cómo se puede observar en la columna “C” la utilización del cpu es de 0 exceptuando en 1 sola columna. Por ende es bastante deficiente ya que puede ser que algunos threads utilicen más o menos recursos que otros y entonces muchos threads no se les asigne toda su utilización.

10. ¿Cuál es el efecto de agregar omp_set_nested(true)? Explique
    Lo que permite ques que se pueda realizar la paralelización anidada, es decir cuando un thread entra en una región paralela y esta región paralela pueda crear otras regiones paralelas, es paralelizar lo paralelizado.