Universidad del Valle De Guatemala

Facultad de Ingeniería

Sistemas Operativos

Lab 3

Javier Mombiela
Carne: 20067
Sección: 21

Guatemala, 8 de marzo 2023

Cree un programa en C llamado SudokuValidator.c. En él escriba tres funciones que se encarguen de revisar que todos los números del uno al nueve estén:
• En cada columna de un arreglo de nueve por nueve.
• En cada fila de un arreglo de nueve por nueve.
• En un subarreglo de tres por tres dentro de un arreglo de nueve por nueve.

Las funciones para verificación de filas y columnas serán iguales exceptuando un intercambio de índices al recorrer el arreglo. La función de revisión de subarreglos debe recibir una fila y una columna para ubicar la esquina superior izquierda de un cuadrado de tres por tres, donde iniciará la revisión dentro del arreglo de nueve por nueve. Todas estas funciones se deben basar en ciclos for obligatoriamente.

Este programa recibirá, en terminal, la ubicación de un archivo (sólo el nombre, si está en el mismo directorio que SudokuValidator.c) que contiene una solución a un sudoku de nueve por nueve. El formato de las soluciones debe ser un único string de ochenta y un dígitos, en la primera línea, comenzando por la celda superior izquierda del sudoku y avanzando de izquierda a derecha, por filas. Para este laboratorio se provee una solución de ejemplo en el archivo “sudoku”.

Lo primero que su main() deberá hacer es abrir el archivo usando open() y mappearlo a su memoria usando mmap(). Luego debe ejecutar un for en el que se copie cada símbolo del string en el archivo de solución a un arreglo bidimensional de nueve por nueve, de modo que le quede una grilla lógica como la que se muestra en la página siguiente.

Se recomienda que su arreglo bidimensional sea global (es decir, que esté declarado fuera del main()) para que sea accesible por varios threads. Luego de llenar la grilla, escriba un for que haga la revisión, con su función, de los subarreglos de tres por tres que conforman el arreglo de nueve por nueve (nota:revise los subarreglos de tres por tres cuya primera posición (si comenzamos desde 1) sea [*ii*, *ii*] para $ii \in \{1,4,7\}$).

```
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$ ./lab3 sudoku

Arreglo bidimensional de 9x9:
6 2 4 5 3 9 1 8 7
5 1 9 7 2 8 6 3 4
8 3 7 6 1 4 2 9 5
1 4 3 8 6 5 7 2 9
9 5 8 2 4 7 3 6 1
7 6 2 3 9 1 4 5 8
3 7 1 9 5 6 8 4 2
4 9 6 1 8 2 5 7 3
2 8 5 4 7 3 9 1 6

Subarreglo de 1x1 -> correcto
Subarreglo de 1x4 -> correcto
Subarreglo de 1x7 -> correcto
Subarreglo de 4x1 -> correcto
Subarreglo de 4x4 -> correcto
Subarreglo de 4x7 -> correcto
Subarreglo de 7x1 -> correcto
Subarreglo de 7x4 -> correcto
Subarreglo de 7x7 -> correcto
```

Luego de lo anterior, obtenga el número de proceso (no el de thread) y ejecute un fork(). En el proceso hijo convierta el número del proceso padre (no el de thread) a texto, y ejecute por medio de execlp() el siguiente comando:

```
ps -p <#proc> -lLf
```

donde <#proc> es el número del proceso padre. Este comando permite ver información relacionada al proceso <#proc> que incluye los lightweight processes que tenga asociados.

En el proceso padre:
- Cree un pthread que haga su revisión de columnas.
- Ejecute pthread_join() y luego despliegue el número de thread en ejecución. Para lograrlo debe #incluir en su programa y ejecutar syscall(SYS_gettid) (el resultado de esta llamada de sistema es el id del thread).
- Espere a que concluya el hijo que está ejecutando ps.
- Realice su revisión de filas. - Despliegue si la solución al sudoku es válida o no.
- Ejecute un nuevo fork() y ejecute el comando ps en el proceso hijo, tal como se describe en instrucciones anteriores. Esto servirá para comparar el número de LWP's asociados al proceso padre cuando se está realizando la revisión de columnas y cuando (el padre) está a punto de terminar.
- Espere al hijo y retorne 0.

Observe que la creación de un thread que ejecute la revisión de columnas implica la creación de una función que sea asignable a un thread en el cual, a su vez, se ejecute su función de revisión de columnas. Es decir, una función que tenga tipo de retorno void* y que termine con pthreads_exit(0). En esa función tipo void* también despliegue el número de thread en ejecución.

```
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$ ./lab3 sudoku
El thread que ejecuta el metodo para ejecutar el metodo de revision de columnas es: 6023
En la revision de columnas el siguiente es un thread en ejecucion: 6023
En la revision de columnas el siguiente es un thread en ejecucion: 6023
En la revision de columnas el siguiente es un thread en ejecucion: 6023
En la revision de columnas el siguiente es un thread en ejecucion: 6023
En la revision de columnas el siguiente es un thread en ejecucion: 6023
En la revision de columnas el siguiente es un thread en ejecucion: 6023
En la revision de columnas el siguiente es un thread en ejecucion: 6023
En la revision de columnas el siguiente es un thread en ejecucion: 6023
En la revision de columnas el siguiente es un thread en ejecucion: 6023
El thread en que se ejecuta en el main es: 6021
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    6021    1789    6021  0    1  80   0 - 19158 do_wai 13:48 pts/0    00:00:00 ./lab3

El sudoku SI es válido!
Antes de terminar el estado de este proceso y sus threads es:
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    6021    1789    6021  0    1  80   0 - 19158 do_wai 13:48 pts/0    00:00:00 ./lab3
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$
```

Como siguiente paso deberá paralelizar todos los ciclos for que pueda (vea la nota importante) usando OpenMP. Para ello simplemente es necesario que la siguiente línea preceda inmediatamente a la del for en cada caso:

```
#pragma omp parallel for
```

Importante: evite las race conditions. Investigue el uso de la directiva private de OpenMP para auxiliarse en este aspecto. No todos los ciclos for deberán ser precedidos por la directiva.

- Una race condition se produce cuando dos o más hilos intentan acceder y modificar la misma variable o recurso al mismo tiempo, lo que puede dar lugar a resultados impredecibles y errores en el programa.

- Para evitar una race condition, se puede utilizar la directiva "private" de OpenMP, que permite especificar que una variable debe ser privada para cada hilo y no compartida entre ellos. Esto significa que cada hilo tendrá su propia copia de la variable, lo que evitará que los hilos interfieran entre sí.

```
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$ ./lab3 sudoku
El thread que ejecuta el metodo para ejecutar el metodo de revision de columnas es: 7632
En la revision de columnas el siguiente es un thread en ejecucion: 7633
En la revision de columnas el siguiente es un thread en ejecucion: 7635
En la revision de columnas el siguiente es un thread en ejecucion: 7633
En la revision de columnas el siguiente es un thread en ejecucion: 7635
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    7630    1789    7630  0    5  80   0 - 58116 futex_ 14:28 pts/0    00:00:00 ./lab3
En la revision de columnas el siguiente es un thread en ejecucion: 7634
1 S javimom+    7630    1789    7632  0    5  80   0 - 58116 futex_ 14:28 pts/0    00:00:00 ./lab3
1 R javimom+    7630    1789    7633  0    5  80   0 - 58116 -      14:28 pts/0    00:00:00 ./lab3
1 R javimom+    7630    1789    7634  0    5  80   0 - 74500 -      14:28 pts/0    00:00:00 ./lab3
En la revision de columnas el siguiente es un thread en ejecucion: 7634
1 R javimom+    7630    1789    7635  0    5  80   0 - 74500 -      14:28 pts/0    00:00:00 ./lab3
En la revision de columnas el siguiente es un thread en ejecucion: 7632
En la revision de columnas el siguiente es un thread en ejecucion: 7632
En la revision de columnas el siguiente es un thread en ejecucion: 7632
El thread en que se ejecuta en el main es: 7630

El sudoku SI es válido!
Antes de terminar el estado de este proceso y sus threads es:
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    7630    1789    7630  0    4  80   0 - 74532 do_wai 14:28 pts/0    00:00:00 ./lab3
1 R javimom+    7630    1789    7636  0    4  80   0 - 74532 -      14:28 pts/0    00:00:00 ./lab3
1 R javimom+    7630    1789    7637  0    4  80   0 - 74532 -      14:28 pts/0    00:00:00 ./lab3
1 R javimom+    7630    1789    7638  0    4  80   0 - 74532 -      14:28 pts/0    00:00:00 ./lab3
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$
```

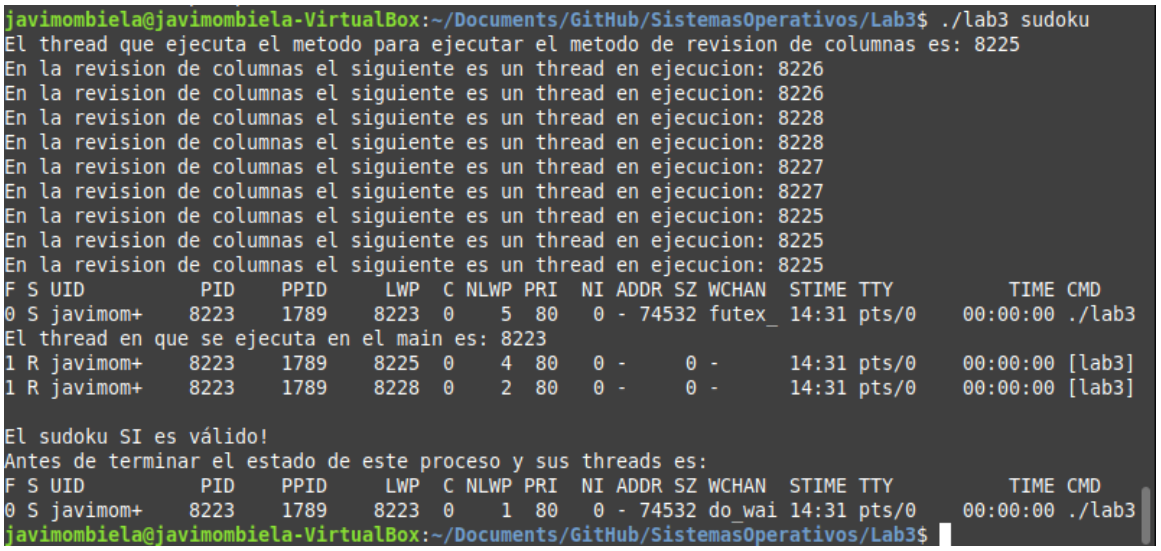
Anote el número de LWP’s que se tienen durante la revisión de columnas y antes de terminar el programa.

- El número de LWP’s que se tiene durante la revision es: 5
- El número de LWP’s que se tiene antes de terminar el programa es: 4

Agregue la siguiente instrucción al principio de main():

```
omp_set_num_threads(1);
```

Ejecute su programa y note el resultado de las ejecuciones de ps. También anote los números de thread desplegados durante la revisión de columnas.



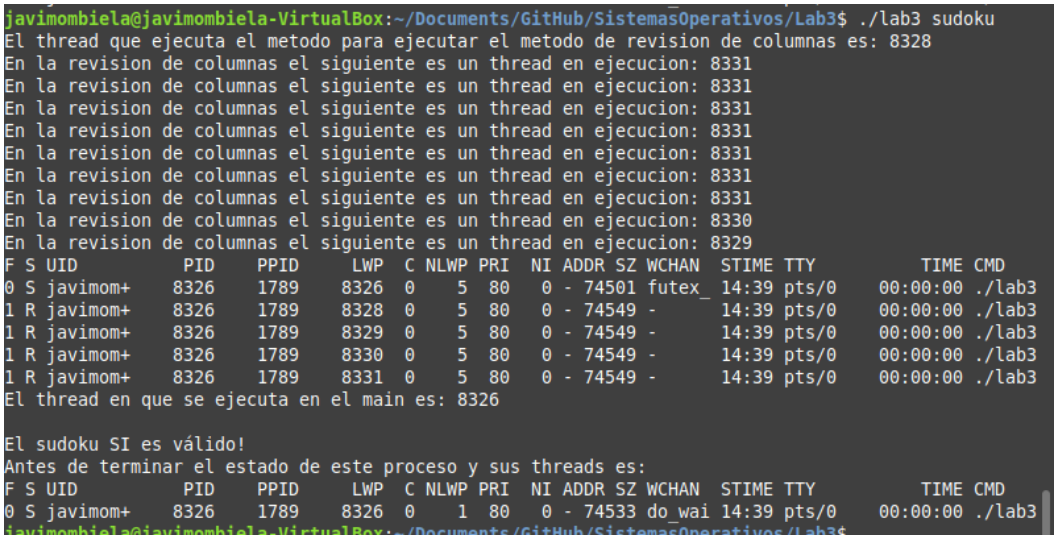
```
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$ ./lab3 sudoku
El thread que ejecuta el metodo para ejecutar el metodo de revision de columnas es: 8225
En la revision de columnas el siguiente es un thread en ejecucion: 8226
En la revision de columnas el siguiente es un thread en ejecucion: 8226
En la revision de columnas el siguiente es un thread en ejecucion: 8228
En la revision de columnas el siguiente es un thread en ejecucion: 8228
En la revision de columnas el siguiente es un thread en ejecucion: 8227
En la revision de columnas el siguiente es un thread en ejecucion: 8227
En la revision de columnas el siguiente es un thread en ejecucion: 8225
En la revision de columnas el siguiente es un thread en ejecucion: 8225
En la revision de columnas el siguiente es un thread en ejecucion: 8225
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    8223    1789    8223  0    5  80   0 - 74532 futex_ 14:31 pts/0    00:00:00 ./lab3
El thread en que se ejecuta en el main es: 8223
1 R javimom+    8223    1789    8225  0    4  80   0 -     0 -      14:31 pts/0    00:00:00 [lab3]
1 R javimom+    8223    1789    8228  0    2  80   0 -     0 -      14:31 pts/0    00:00:00 [lab3]

El sudoku SI es válido!
Antes de terminar el estado de este proceso y sus threads es:
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    8223    1789    8223  0    1  80   0 - 74532 do_wai 14:31 pts/0    00:00:00 ./lab3
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$
```

Numeros de threads:
8225
8226
8227
8228

Ahora, agregue la siguiente directiva a todas las líneas #pragma que incluyó anteriormente: `schedule(dynamic)`

Ejecute su programa varias veces y observe los números de thread que se despliegan durante la revisión de columnas. Compárelos con el resultado de ps que se despliega durante la ejecución del pthread y anote sus observaciones.



```
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$ ./lab3 sudoku
El thread que ejecuta el metodo para ejecutar el metodo de revision de columnas es: 8328
En la revision de columnas el siguiente es un thread en ejecucion: 8331
En la revision de columnas el siguiente es un thread en ejecucion: 8331
En la revision de columnas el siguiente es un thread en ejecucion: 8331
En la revision de columnas el siguiente es un thread en ejecucion: 8331
En la revision de columnas el siguiente es un thread en ejecucion: 8331
En la revision de columnas el siguiente es un thread en ejecucion: 8331
En la revision de columnas el siguiente es un thread en ejecucion: 8331
En la revision de columnas el siguiente es un thread en ejecucion: 8330
En la revision de columnas el siguiente es un thread en ejecucion: 8329
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    8326    1789    8326  0    5  80   0 - 74501 futex_ 14:39 pts/0    00:00:00 ./lab3
1 R javimom+    8326    1789    8328  0    5  80   0 - 74549 -      14:39 pts/0    00:00:00 ./lab3
1 R javimom+    8326    1789    8329  0    5  80   0 - 74549 -      14:39 pts/0    00:00:00 ./lab3
1 R javimom+    8326    1789    8330  0    5  80   0 - 74549 -      14:39 pts/0    00:00:00 ./lab3
1 R javimom+    8326    1789    8331  0    5  80   0 - 74549 -      14:39 pts/0    00:00:00 ./lab3
El thread en que se ejecuta en el main es: 8326

El sudoku SI es válido!
Antes de terminar el estado de este proceso y sus threads es:
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    8326    1789    8326  0    1  80   0 - 74533 do_wai 14:39 pts/0    00:00:00 ./lab3
```

Como siguiente paso, agregue una llamada a omp_set_num_threads() al inicio de cada función donde se ejecute un for paralelo, determinando el número de threads adecuados (e.g., si su función ejecuta un for paralelo de nueve iteraciones, posiblemente el número de threads deba ser nueve). Ejecute su programa varias veces y anote los efectos sobre los threads en los resultados de ps. Repita el procedimiento comentando la cláusula schedule() en el primer for paralelo de su revisión de columnas. Finalmente agregue la siguiente instrucción al principio de cada función que use OpenMP:

```
omp_set_nested(true);
```

Ejecute su programa y anote los efectos sobre el resultado.

```
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$ ./lab3 sudoku
El thread que ejecuta el metodo para ejecutar el metodo de revision de columnas es: 8446
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    8388    1789    8388  0   13  80   0 - 74509 futex_ 14:42 pts/0    00:00:00 ./lab3
1 R javimom+    8388    1789    8389  0   12  80   0 - 74509 -      14:42 pts/0    00:00:00 ./lab3
1 R javimom+    8388    1789    8390  0   12  80   0 - 74509 -      14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8446  0   12  80   0 - 74509 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8447  0   12  80   0 - 74509 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8448  0   12  80   0 - 74509 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8449  0   12  80   0 - 74509 futex_ 14:42 pts/0    00:00:00 ./lab3
1 R javimom+    8388    1789    8450  0   12  80   0 - 74509 -      14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8451  0   12  80   0 - 74509 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8452  0   12  80   0 - 74509 futex_ 14:42 pts/0    00:00:00 ./lab3
1 R javimom+    8388    1789    8453  0   12  80   0 - 74509 -      14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8454  0   12  80   0 - 74509 futex_ 14:42 pts/0    00:00:00 ./lab3
En la revision de columnas el siguiente es un thread en ejecucion: 8449
En la revision de columnas el siguiente es un thread en ejecucion: 8446
En la revision de columnas el siguiente es un thread en ejecucion: 8451
En la revision de columnas el siguiente es un thread en ejecucion: 8454
En la revision de columnas el siguiente es un thread en ejecucion: 8448
En la revision de columnas el siguiente es un thread en ejecucion: 8447
En la revision de columnas el siguiente es un thread en ejecucion: 8452
En la revision de columnas el siguiente es un thread en ejecucion: 8450
En la revision de columnas el siguiente es un thread en ejecucion: 8453
El thread en que se ejecuta en el main es: 8388

El sudoku SI es válido!
Antes de terminar el estado de este proceso y sus threads es:
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S javimom+    8388    1789    8388  0    9  80   0 - 205613 do_wai 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8389  0    9  80   0 - 205613 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8390  0    9  80   0 - 205613 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8527  0    9  80   0 - 205613 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8528  0    9  80   0 - 205613 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8529  0    9  80   0 - 205613 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8530  0    9  80   0 - 205613 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8531  0    9  80   0 - 205613 futex_ 14:42 pts/0    00:00:00 ./lab3
1 S javimom+    8388    1789    8532  0    9  80   0 - 205613 futex_ 14:42 pts/0    00:00:00 ./lab3
javimombiela@javimombiela-VirtualBox:~/Documents/GitHub/SistemasOperativos/Lab3$
```

1. **¿Qué es una race condition y por qué hay que evitarlas?**
    a. Una race condition es un problema que ocurre cuando dos o más procesos compiten por acceder a un recurso compartido y el resultado depende del orden de ejecución. Es importante evitar una race condition porque puede causar errores impredecibles en el sistema, pérdida de datos y problemas de seguridad. Una race condition ocurre cuando dos o más procesos compiten por

el mismo recurso compartido y el resultado final depende del orden de ejecución

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?**
    a. En Linux, pthreads es una forma más fácil y conveniente de crear y administrar hilos en un programa que clone(). Aunque pthreads utiliza internamente la llamada al sistema clone() para crear hilos, pthreads ofrece una interfaz de programación más amigable y características adicionales como sincronización y comunicación entre hilos. En general, se recomienda utilizar pthreads para la creación de hilos en Linux, a menos que se necesite un control más preciso sobre los recursos del sistema, en cuyo caso clone() puede ser utilizado directamente.
3. **¿Dónde, en su programa, hay paralelización de tareas, y dónde de datos?**
    a. En el programa, podemos observar que hay paralelización de tareas en cada una de las funciones que verifican si el sudoku es válido o no. Esto se hace al momento de utilizar la librería OpenMP en cada una de estas funciones. Por el otro lado, podemos ver que la paralelización de datos se hace en la parte en donde se crean los threads para poder verificar las columnas del sudoku y poder hacerlo en varios hilos.
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 abiertos en cada caso, entonces? Hint: recuerde el modelo de multithreading que usan Linux y Windows.**
    a. El número de LWP's que están abiertos antes de terminar el main() son 4, y el número durante la revisión de columnas son 5. Entonces podemos decir que el primer caso deberia haber al menos 4 y en el segundo caso debe haber al menos 5.
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?**
    a. El número de LWP's que están abiertos antes de terminar el main() es 1 Esto se debe a que OpenMP solo está utilizando un hilo en lugar de varios hilos para la ejecución paralela. El número durante la revisión de columnas son 3. Por defecto, OpenMP crea un número de hilos igual al número de núcleos disponibles en la CPU. Sin embargo, este número puede ser cambiado con la llamada a la función omp_set_num_threads().
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.**
    a. La primera columna de los resultados del comando ps es el ID de proceso (PID), que es un identificador único asignado a cada proceso en el sistema operativo. En cuanto al LWP inactivo, es posible que sea el LWP que se creó

para el hilo principal de ejecución y que no tenga trabajo asignado en ese momento. También es posible que esté esperando algún recurso o que esté bloqueado por algún motivo.

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?**
    a. En la pregunta anterior, se observó que durante la revisión de columnas, hay varios LWP's abiertos según el comando ps. Estos LWP's corresponden a los threads creados por OpenMP para ejecutar la tarea de manera paralela.
    b. En OpenMP, un grupo de threads se conoce como un "thread team", que se crea al inicio de una sección paralela y se destruye al finalizar la sección. Dentro de un thread team, hay un thread principal, conocido como "master thread", que es el encargado de dividir el trabajo entre los demás threads.
    c. En la función de revisión de columnas, el thread "corriendo" pero que no está haciendo nada es el thread principal o "master thread". Esto sucede porque el thread principal se encarga de dividir el trabajo entre los demás threads, pero como en este caso se limitó el número de threads a uno, no hay otros threads disponibles para ejecutar el trabajo. Por lo tanto, el thread principal debe esperar a que el otro thread finalice para continuar con la ejecución.
    d. El término "busy-wait" se refiere a una técnica en la que un thread espera activamente (es decir, realiza un ciclo continuo) hasta que se cumple una cierta condición. En este caso, el thread principal está esperando activamente a que el otro thread termine de ejecutar su tarea.
    e. OpenMP maneja su pool de threads de manera dinámica. Por defecto, OpenMP crea un número de threads igual al número de núcleos de la CPU disponible. Sin embargo, este número se puede ajustar mediante la llamada a la función omp_set_num_threads(). Durante la ejecución del programa, los threads se crean y destruyen según sea necesario para ejecutar la tarea de manera paralela.
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?**
    a. La cantidad máxima de threads trabajando luego de agregar la cláusula schedule(dynamic) es igual al número de columnas en el tablero de ajedrez. Esto sugiere que por defecto OpenMP distribuye el trabajo de manera equitativa entre los threads disponibles, lo que no necesariamente garantiza la máxima eficiencia en términos de utilización de recursos.
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.

a. A la hora de agregar la llamada omp_set_num_threads() podemos observar que en el programa, hay más concurrencia, ya que ahora, gracias a estas llamadas, se pueden realizar varias tareas al mismo tiempo.

10. **¿Cuál es el efecto de agregar omp_set_nested(true)? Explique**
    a. La función omp_set_nested(true) permite que se creen hilos dentro de hilos en un programa que utiliza OpenMP. Esto significa que se pueden crear tareas paralelas más complejas y anidadas, lo que puede mejorar el rendimiento del programa en sistemas multiprocesador. En términos más simples, esto permite que el programa use mejor los recursos de la computadora para realizar varias tareas al mismo tiempo y de manera más eficiente.