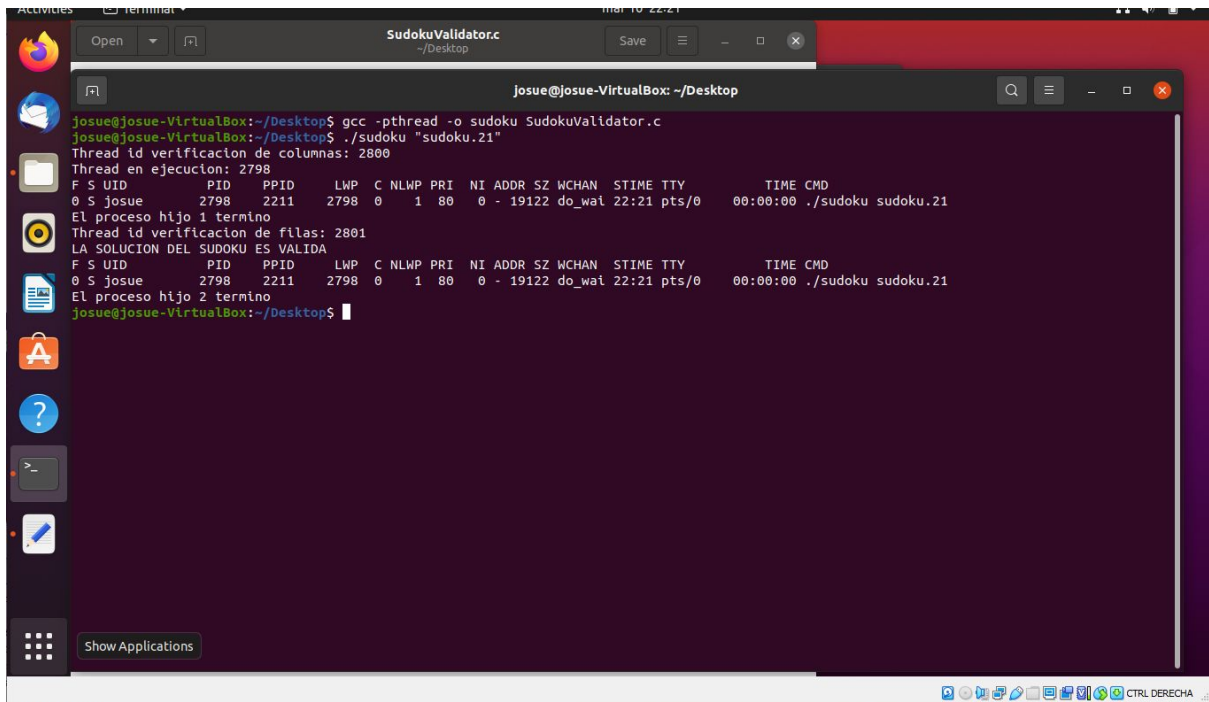


Josué Sagastume - 18173
Sistemas Operativos
LABORATORIO 3

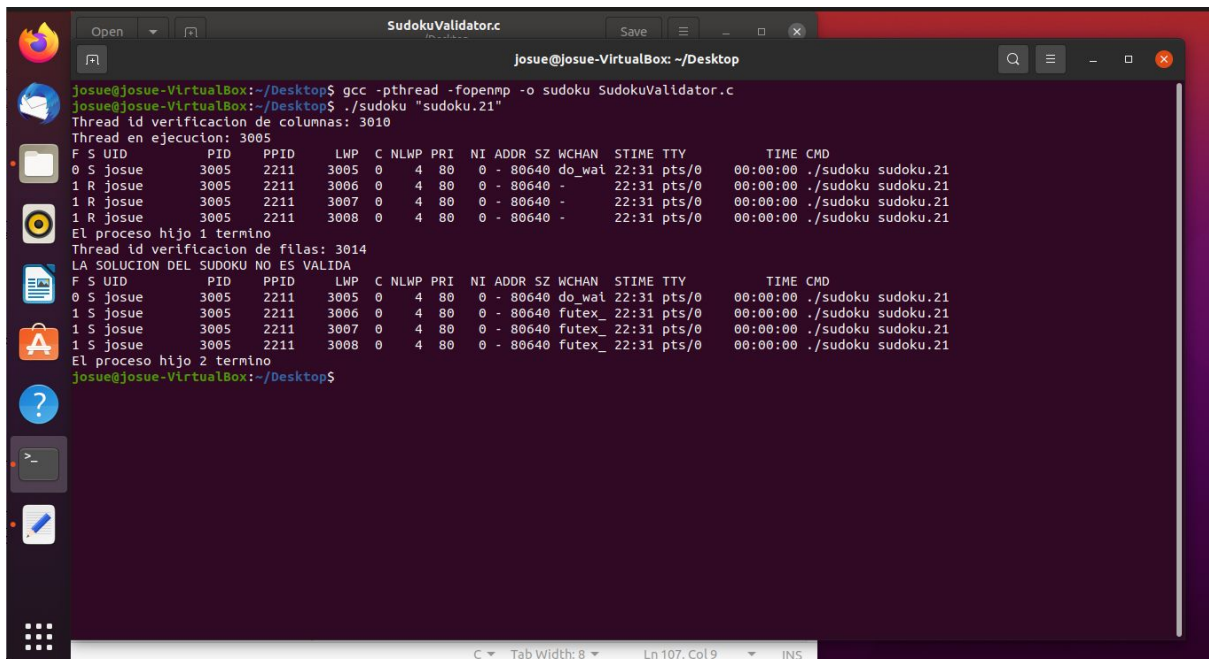
Ejercicio 1



```

Josue@Josue-VirtualBox: ~/Desktop
$ gcc -pthread -o sudoku SudokuValidator.c
$ ./sudoku "sudoku.21"
Thread id verificacion de columnas: 2800
Thread en ejecucion: 2798
F S UID      PID      PPID      LWP      C  NLWP  PRI  NI ADDR SZ  WCHAN  STIME TTY      TIME CMD
0 S josue    2798     2211    2798    0   1    80   0 - 19122 do_wai 22:21 pts/0    00:00:00 ./sudoku sudoku.21
El proceso hijo 1 termino
Thread id verificacion de filas: 2801
LA SOLUCION DEL SUDOKU ES VALIDA
F S UID      PID      PPID      LWP      C  NLWP  PRI  NI ADDR SZ  WCHAN  STIME TTY      TIME CMD
0 S josue    2798     2211    2798    0   1    80   0 - 19122 do_wai 22:21 pts/0    00:00:00 ./sudoku sudoku.21
El proceso hijo 2 termino
Josue@Josue-VirtualBox:~/Desktop$
```

Imagen 1. Primera ejecución del programa.



```

Josue@Josue-VirtualBox: ~/Desktop
$ gcc -pthread -fopenmp -o sudoku SudokuValidator.c
$ ./sudoku "sudoku.21"
Thread id verificacion de columnas: 3010
Thread en ejecucion: 3005
F S UID      PID      PPID      LWP      C  NLWP  PRI  NI ADDR SZ  WCHAN  STIME TTY      TIME CMD
0 S josue    3005     2211    3005    0   4    80   0 - 80640 do_wai 22:31 pts/0    00:00:00 ./sudoku sudoku.21
1 R josue    3005     2211    3006    0   4    80   0 - 80640 -      22:31 pts/0    00:00:00 ./sudoku sudoku.21
1 R josue    3005     2211    3007    0   4    80   0 - 80640 -      22:31 pts/0    00:00:00 ./sudoku sudoku.21
1 R josue    3005     2211    3008    0   4    80   0 - 80640 -      22:31 pts/0    00:00:00 ./sudoku sudoku.21
El proceso hijo 1 termino
Thread id verificacion de filas: 3014
LA SOLUCION DEL SUDOKU NO ES VALIDA
F S UID      PID      PPID      LWP      C  NLWP  PRI  NI ADDR SZ  WCHAN  STIME TTY      TIME CMD
0 S josue    3005     2211    3005    0   4    80   0 - 80640 do_wai 22:31 pts/0    00:00:00 ./sudoku sudoku.21
1 S josue    3005     2211    3006    0   4    80   0 - 80640 futex_ 22:31 pts/0    00:00:00 ./sudoku sudoku.21
1 S josue    3005     2211    3007    0   4    80   0 - 80640 futex_ 22:31 pts/0    00:00:00 ./sudoku sudoku.21
1 S josue    3005     2211    3008    0   4    80   0 - 80640 futex_ 22:31 pts/0    00:00:00 ./sudoku sudoku.21
El proceso hijo 2 termino
Josue@Josue-VirtualBox:~/Desktop$
```

Imagen 2. Ejecución del programa con el comando *#pragma omp parallel for*.

```

Activities Terminal mar 10 22:34
SudokuValidator.c
josue@josue-VirtualBox: ~/Desktop
josue@josue-VirtualBox:~/Desktop$ gcc -pthread -fopenmp -o sudoku SudokuValidator.c
josue@josue-VirtualBox:~/Desktop$ ./sudoku "sudoku.21"
Thread id verificacion de columnas: 3064
F S UID PID PPID LWP C NLWP PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
0 S josue 3062 2211 3062 0 5 80 0 - 25341 futex_ 22:34 pts/0 00:00:00 ./sudoku sudoku.21
1 R josue 3062 2211 3064 0 2 80 0 - 25341 - 22:34 pts/0 00:00:00 ./sudoku sudoku.21
Thread en ejecucion: 3062
El proceso hijo 1 termino
Thread id verificacion de filas: 3068
LA SOLUCION DEL SUDOKU ES VALIDA
F S UID PID PPID LWP C NLWP PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
0 S josue 3062 2211 3062 0 1 80 0 - 25341 do_wai 22:34 pts/0 00:00:00 ./sudoku sudoku.21
El proceso hijo 2 termino
josue@josue-VirtualBox:~/Desktop$

```

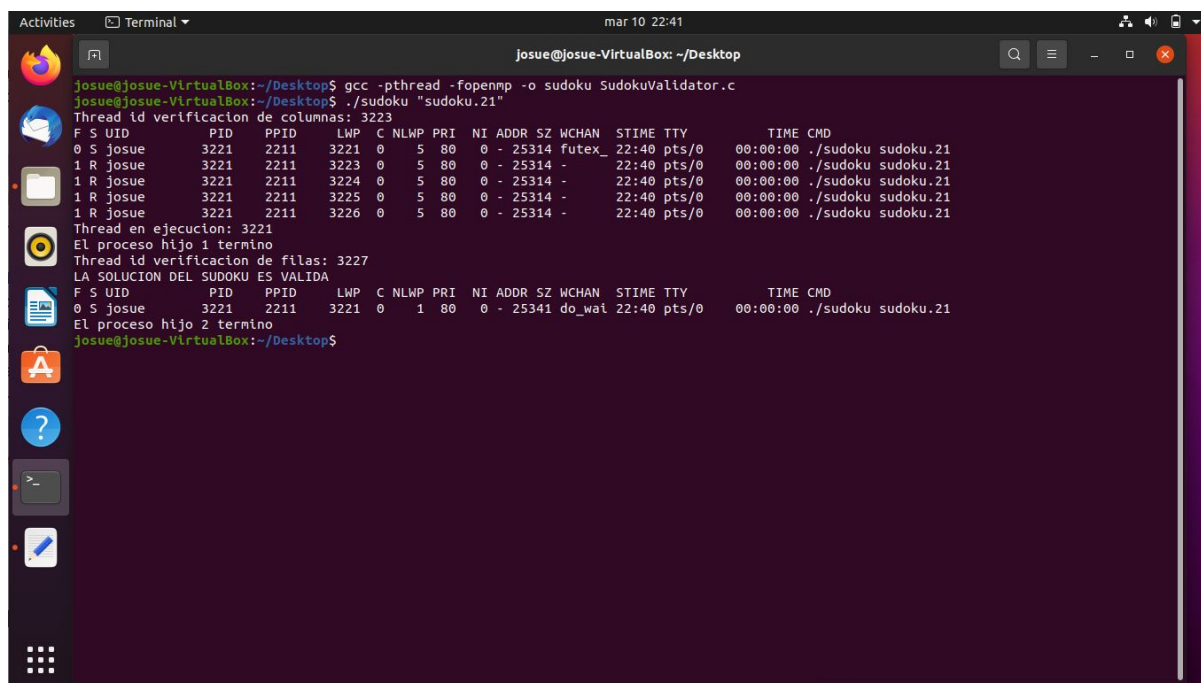
Imagen 3. Ejecución del programa con la instrucción `omp_set_num_threads(1)` al inicio del `main()`.

```

josue@josue-VirtualBox: ~/Desktop
josue@josue-VirtualBox:~/Desktop$ gcc -pthread -fopenmp -o sudoku SudokuValidator.c
josue@josue-VirtualBox:~/Desktop$ ./sudoku "sudoku.21"
Thread id verificacion de columnas: 3148
F S UID PID PPID LWP C NLWP PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
0 S josue 3146 2211 3146 0 5 80 0 - 25314 futex_ 22:37 pts/0 00:00:00 ./sudoku sudoku.21
1 R josue 3146 2211 3148 0 5 80 0 - 25314 - 22:37 pts/0 00:00:00 ./sudoku sudoku.21
1 R josue 3146 2211 3149 0 5 80 0 - 25314 - 22:37 pts/0 00:00:00 ./sudoku sudoku.21
1 R josue 3146 2211 3150 0 5 80 0 - 25314 - 22:37 pts/0 00:00:00 ./sudoku sudoku.21
1 R josue 3146 2211 3151 0 5 80 0 - 25314 - 22:37 pts/0 00:00:00 ./sudoku sudoku.21
Thread en ejecucion: 3146
El proceso hijo 1 termino
Thread id verificacion de filas: 3152
LA SOLUCION DEL SUDOKU ES VALIDA
F S UID PID PPID LWP C NLWP PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
0 S josue 3146 2211 3146 0 1 80 0 - 25341 do_wai 22:37 pts/0 00:00:00 ./sudoku sudoku.21
El proceso hijo 2 termino
josue@josue-VirtualBox:~/Desktop$ ./sudoku "sudoku.21"
Thread id verificacion de columnas: 3160
Thread en ejecucion: 3158
F S UID PID PPID LWP C NLWP PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
0 S josue 3158 2211 3158 0 1 80 0 - 25341 do_wai 22:37 pts/0 00:00:00 ./sudoku sudoku.21
El proceso hijo 1 termino
Thread id verificacion de filas: 3164
LA SOLUCION DEL SUDOKU ES VALIDA
F S UID PID PPID LWP C NLWP PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
0 S josue 3158 2211 3158 0 1 80 0 - 25341 do_wai 22:37 pts/0 00:00:00 ./sudoku sudoku.21
El proceso hijo 2 termino
josue@josue-VirtualBox:~/Desktop$ ./sudoku "sudoku.21"
Thread id verificacion de columnas: 3171
F S UID PID PPID LWP C NLWP PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
0 S josue 3169 2211 3169 0 5 80 0 - 25334 futex_ 22:37 pts/0 00:00:00 ./sudoku sudoku.21
1 R josue 3169 2211 3171 0 5 80 0 - 25361 - 22:37 pts/0 00:00:00 ./sudoku sudoku.21
1 R josue 3169 2211 3172 0 5 80 0 - 25361 - 22:37 pts/0 00:00:00 ./sudoku sudoku.21
1 R josue 3169 2211 3173 0 5 80 0 - 25361 - 22:37 pts/0 00:00:00 ./sudoku sudoku.21
1 R josue 3169 2211 3174 0 5 80 0 - 25361 - 22:37 pts/0 00:00:00 ./sudoku sudoku.21
Thread en ejecucion: 3169
El proceso hijo 1 termino
Thread id verificacion de filas: 3175

```

Imagen 4. Ejecución del programa agregando la directiva `schedule(dynamic)`.



```
Activities Terminal mar 10 22:41
josue@josue-VirtualBox: ~/Desktop
josue@josue-VirtualBox:~/Desktop$ gcc -pthread -fopenmp -o sudoku SudokuValidator.c
josue@josue-VirtualBox:~/Desktop$ ./sudoku "sudoku.21"
Thread id verification de columnas: 3223
F S UID      PID      PPID      LWP      C  NLWP  PRI  NI ADDR SZ  WCHAN    STIME TTY      TIME CMD
0 S josue    3221     2211     3221    0   5    80   0 - 25314 futex_ 22:40 pts/0    00:00:00 ./sudoku sudoku.21
1 R josue    3221     2211     3223    0   5    80   0 - 25314 -      22:40 pts/0    00:00:00 ./sudoku sudoku.21
1 R josue    3221     2211     3224    0   5    80   0 - 25314 -      22:40 pts/0    00:00:00 ./sudoku sudoku.21
1 R josue    3221     2211     3225    0   5    80   0 - 25314 -      22:40 pts/0    00:00:00 ./sudoku sudoku.21
1 R josue    3221     2211     3226    0   5    80   0 - 25314 -      22:40 pts/0    00:00:00 ./sudoku sudoku.21
Thread en ejecución: 3221
El proceso hijo 1 termino
Thread id verification de filas: 3227
LA SOLUCION DEL SUDOKU ES VALIDA
F S UID      PID      PPID      LWP      C  NLWP  PRI  NI ADDR SZ  WCHAN    STIME TTY      TIME CMD
0 S josue    3221     2211     3221    0   1    80   0 - 25341 do_wai 22:40 pts/0    00:00:00 ./sudoku sudoku.21
El proceso hijo 2 termino
josue@josue-VirtualBox:~/Desktop$
```

Imagen 5. Ejecución del programa agregando la instrucción `omp_set_nested(true)`.

Responda las siguientes preguntas:

1. ¿Qué es una *race condition* y por qué hay que evitarlas?

Es cuando muchos procesos acceden y manipulan los mismos datos al mismo tiempo y el resultado de la ejecución depende del orden particular en el que tiene lugar el acceso. Debemos de evitar esta situación, pues estos procesos no siempre van a ejecutarse en el mismo orden, dándonos un resultado diferente al que posiblemente esperábamos, y probablemente generando uno que otro error.

2. ¿Cuál es la relación, en Linux, entre *threads* y `clone()`? ¿Hay diferencias al crear *threads* con uno o con otro? ¿Qué es más recomendable?

Linux provee la habilidad de crear *threads* utilizando la llamada al sistema `clone()`, aunque en realidad, Linux no distingue entre procesos y *threads*, pues utiliza el término tarea para referirse a ambas cuando se refiere a un flujo de control en el programa. Cuando se utiliza `clone()`, a este se le pasa un conjunto de indicadores que determinan cuánto se debe compartir entre las tareas principales y secundarias. Esto sería un equivalente a crear un *thread*. Para la creación de este basta con realizar la llamada al sistema `clone()`.

Los *threads* hacen referencia al estándar POSIX que define una API para la creación y sincronización de *threads*. Esta es una especificación del comportamiento de un *thread*, no una implementación. Para crear un *pthread* primero se debe declarar una variable tipo `pthread_t` y luego llamar la función `pthread_create()` que toma 4 argumentos, el puntero al `thread_id`, atributos, el nombre de la función a ejecutar por el *thread* creado y el último parámetro es para pasarle argumentos a la función.

3. ¿Dónde, en su programa, hay paralelización de tareas, y dónde de datos?

Para este caso, cuando se crean los *threads* hay paralelismo de tareas, pues cada *thread* se encarga de ejecutar una diferente tarea. Y cada vez que utilizamos *omp* se da una paralelización de datos, pues se ejecuta el mismo código en diferentes datos.

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?

Tomando en cuenta el modelo multithreading de Linux, que es de 1 a 1, es decir que mapea cada *thread* a un *kernel thread*. Y tomando en cuenta que *#pragma* agrega un *thread* por cada procesador, en total hay 5 *threads* durante la evaluación y 1 al final.

5. Al limitar el número de *threads* en *main()* a uno, ¿cuántos LWP's hay abiertos durante la revisión de columnas?

Cuando se limitan los *threads* a 1, los LWP, como se puede observar en la imagen 3, disminuyeron a 1.

¿Cuántos *threads* (en general) crea OpenMP por defecto?

Este crea N *threads*, siendo N la cantidad de procesadores de la máquina. Que serían 4 para mi caso, que fue la cantidad de procesadores que le asigné a mi máquina virtual.

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*.

El comando *ps* sirve para mostrar información sobre procesos, en este caso para conocer el estado del proceso se tiene que ver la columna S, los procesos que en esta columna tiene una R es porque están activos (Running) y los que tienen una S es porque están detenidos (Stopped). Para este caso, el proceso que no se está ejecutando es el del *main()*.

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*?

Cuando se utiliza la directiva *omp teams*, esta crea una colección de equipos de *threads*, siendo el *thread master* de cada equipo quien ejecute la región de equipos, que para este caso sería el *thread* que ejecuta los procesos paralelizables.

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 máximo es de 4 *threads*, pues como ya se mencionó anteriormente, *OpenMP* crea por defecto 4 de estos. Se podría deducir que lo que *OpenMP* hace es tratar de distribuir el trabajo, de tal manera que todos los *threads* tengan el mismo 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.**

La concurrencia aumenta, pues al agregar más *threads*, la ejecución de estos va a ser en paralelo. Sin embargo, que haya más *threads* también implica que estos consumen más memoria, más tiempo y más de otros recursos, aumentando el *overhead*. Por lo que no necesariamente va a ser más eficiente.

- 10. ¿Cuál es el efecto de agregar `omp_set_nested(true)`?**

Cuando se evalúa esta rutina con *true*, el valor de ICV `max-active-levels-var` es establecido en el número de niveles activos de paralelismo que admite la implementación, de lo contrario, si es valor de `max-active-levels-var` es mayor que 1, entonces, se establece en 1 y esta rutina queda obsoleta.