Universidad del Valle de Guatemala
Sistemas Operativos
Docentes: Juan Luis Garcia; Juan Carlos Canteo
Semestre I, 2022

Laboratorio #3

AndresEmilioQuintoVillagran 18288

Fecha de Entrega: 23 de marzo, 2022.

Descripción: en este laboratorio se empleará *multithreading* por medio de `pthreads` y OpenMP para desarrollar un verificador de soluciones para *sudokus* de nueve por nueve. Los entregables serán todo el código escrito, así como un documento que responda a las preguntas planteadas al final. Se recomienda auxiliar sus respuestas con *screenshots* de la ejecución de su programa.

Materiales: una máquina virtual con Linux o Windows, pero que tenga GCC, y documentación sobre OpenMP.

Contenido:

OpenMP busca paralelizar (no sólo ejecutar de forma concurrente), por lo que su funcionamiento requiere de más de un procesador. Si usará una máquina virtual, comience por asegurarse de que su máquina cuente con cuatro procesadores (en VirtualBox, ventana *Settings*, menú *System*, *tab Processor*). La cantidad se especifica para uniformizar las respuestas a las preguntas que se plantean al final.

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 $[i, i]$ para $i \in \{1,4,7\}$).

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

Grilla lógica ejemplar

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 `<sys/syscall.h>` 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.

El siguiente es un ejemplo de cómo podría verse el *output* de su programa hasta este momento:

```
os@debian:~/sudoku$ ./SudokuValidator.o sudoku
El thread que ejecuta el metodo para ejecutar el metodo de revision de columnas es: 9027
En la revision de columnas el siguiente es un thread en ejecucion: 9027
En la revision de columnas el siguiente es un thread en ejecucion: 9027
En la revision de columnas el siguiente es un thread en ejecucion: 9027
En la revision de columnas el siguiente es un thread en ejecucion: 9027
En la revision de columnas el siguiente es un thread en ejecucion: 9027
En la revision de columnas el siguiente es un thread en ejecucion: 9027
En la revision de columnas el siguiente es un thread en ejecucion: 9027
En la revision de columnas el siguiente es un thread en ejecucion: 9027
En la revision de columnas el siguiente es un thread en ejecucion: 9027
El thread en el que se ejecuta main es: 9025
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S os        9025  1813  9025  0    1  80   0 -  2842 -      06:53 pts/1    00:00:00 ./SudokuValidator.o sudoku
Sudoku resuelto!
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 os        9025  1813  9025  0    1  80   0 -  2842 -      06:53 pts/1    00:00:00 ./SudokuValidator.o sudoku
os@debian:~/sudoku$
```

R//

```
└─$ ./sudokuSolver "sudoku"
El thread en el que se ejecuta main es: 11961
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 11963
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 11963
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 11963
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 11963
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 11963
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 11963
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 11963
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 11963
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 11963
El thread que ejecuta revision de columnas es: 11961
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S kali       11961    5160   11961  0    1  80   0 - 19131 -      22:44 pts/1    00:00:00 ./sudokuSolver sudoku
La solucion al sudoku NO es valida
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S kali       11961    5160   11961  0    1  80   0 - 19131 -      22:44 pts/1    00:00:00 ./sudokuSolver sudoku

┌──(kali㉿kali)-[~/Desktop/Labs/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:

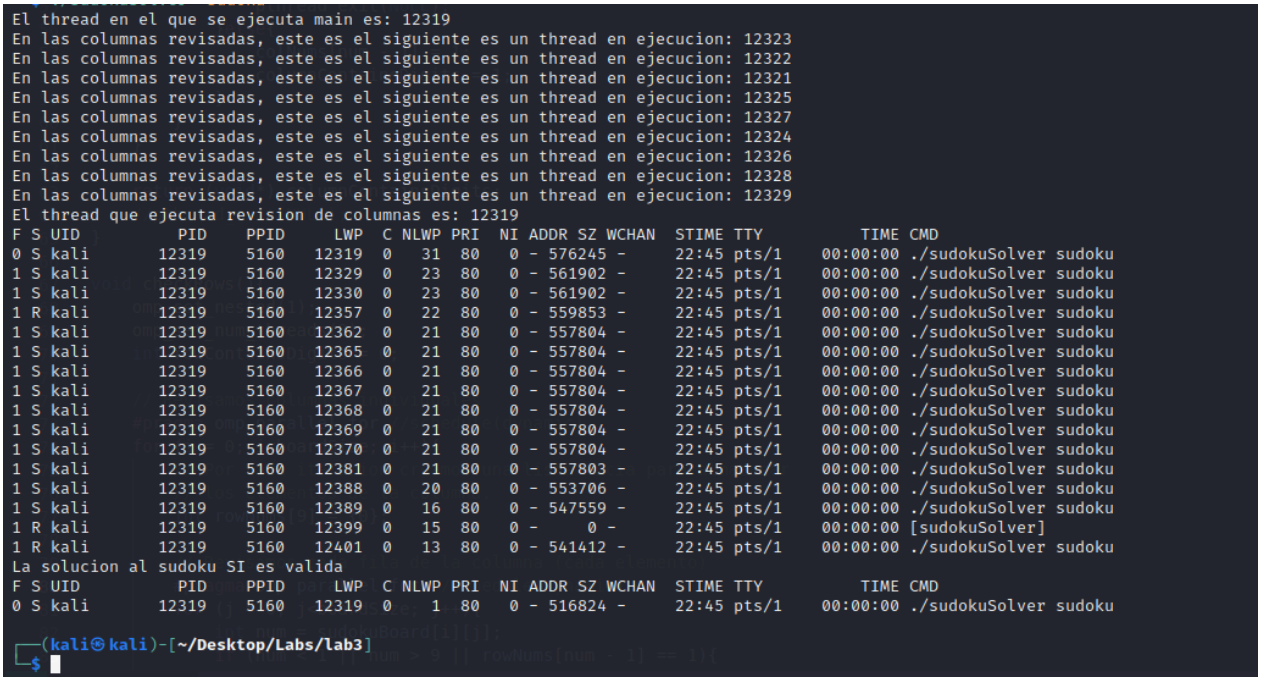
```
#pragma omp parallel for
```

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.

Ejecutar su programa ahora deberá resultar en un *output* similar al siguiente:

```
os@debian:~/sudoku$ ./SudokuValidator.o sudoku
El thread que ejecuta el metodo para ejecutar el metodo de revision de columnas es: 9059
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S os        9054  1813  9054  0    8  80   0 - 15392 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
1 R os        9054  1813  9055  0    8  80   0 - 15392 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
1 R os        9054  1813  9056  0    8  80   0 - 15392 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
1 R os        9054  1813  9057  0    8  80   0 - 15392 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
1 S os        9054  1813  9059  0    8  80   0 - 15392 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
1 S os        9054  1813  9060  0    8  80   0 - 15392 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
En la revision de columnas el siguiente es un thread en ejecucion: 9061
En la revision de columnas el siguiente es un thread en ejecucion: 9061
En la revision de columnas el siguiente es un thread en ejecucion: 9061
1 S os        9054  1813  9061  0    8  80   0 - 15392 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
En la revision de columnas el siguiente es un thread en ejecucion: 9059
En la revision de columnas el siguiente es un thread en ejecucion: 9059
En la revision de columnas el siguiente es un thread en ejecucion: 9059
1 S os        9054  1813  9062  0    8  80   0 - 15392 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
En la revision de columnas el siguiente es un thread en ejecucion: 9060
En la revision de columnas el siguiente es un thread en ejecucion: 9060
En la revision de columnas el siguiente es un thread en ejecucion: 9060
El thread en el que se ejecuta main es: 9054
Sudoku resuelto!
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 os        9054  1813  9054  0    4  80   0 - 15648 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
1 S os        9054  1813  9055  0    4  80   0 - 15648 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
1 S os        9054  1813  9056  0    4  80   0 - 15648 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
1 S os        9054  1813  9057  0    4  80   0 - 15648 -      07:12 pts/1    00:00:00 ./SudokuValidator.o sudoku
os@debian:~/sudoku$
```

R//



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

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.

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

```
schedule(dynamic)
```

```
└─$ ./sudokuSolver "sudoku"
El thread en el que se ejecuta main es: 12722
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 12725
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 12731
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 12729
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 12728
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 12730
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 12732
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 12724
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 12727
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 12726
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S kali       12722    5160   12722  0   15  80   0 - 545528 -     22:46 pts/1    00:00:00 ./sudokuSolver sudoku
1 S kali       12722    5160   12725  0    9  80   0 - 533216 -     22:46 pts/1    00:00:00 ./sudokuSolver sudoku
El thread que ejecuta revision de columnas es: 12722
1 S kali       12722    5160   12726  0    6  80   0 - 529118 -     22:46 pts/1    00:00:00 ./sudokuSolver sudoku
1 R kali       12722    5160   12802  0    2  80   0 - 516824 -     22:46 pts/1    00:00:00 [sudokuSolver]
La solucion al sudoku SI es valida
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S kali       12722    5160   12722  0    1  80   0 - 516824 -     22:46 pts/1    00:00:00 ./sudokuSolver sudoku
```

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.

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.

```
┌──(kali㉿kali)-[~/Desktop/Labs/lab3]
└─$ gcc SudokuValidator.c -o sudokuValidator -fopenmp -lrt
SudokuValidator.c: In function ‘checkColumns’:
SudokuValidator.c:32:9: warning: implicit declaration of function ‘omp_set_nested’ [-Wimplicit-
function-declaration]
```

Universidad del Valle de Guatemala
Sistemas Operativos
Docentes: Juan Luis Garcia; Juan Carlos Canteo
Semestre I, 2022

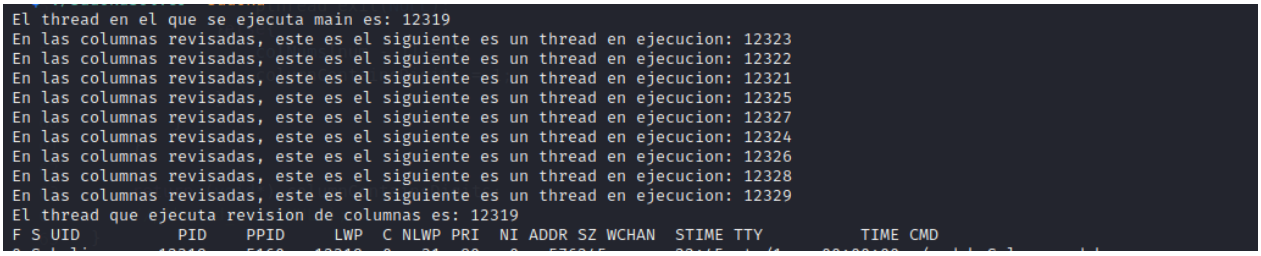
```
┌──(kali㉿kali)-[~/Desktop/Labs/lab3]
└─$ ./sudokuValidator "sudoku"
El thread en el que se ejecuta main es: 5442
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 5445
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 5446
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 5448
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 5449
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 5451
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 5444
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 5450
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 5452
En las columnas revisadas, este es el siguiente es un thread en ejecucion: 5447
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S kali        5442    1136    5442  0   11  80   0 - 539381 -      22:23 pts/0    00:00:00 ./s
1 S kali        5442    1136    5445  0    7  80   0 - 531167 -      22:23 pts/0    00:00:00 ./s
1 R kali        5442    1136    5449  0    7  80   0 - 531167 -      22:23 pts/0    00:00:00 ./s
1 S kali        5442    1136    5451  0    6  80   0 - 529118 -      22:23 pts/0    00:00:00 ./s
1 R kali        5442    1136    5452  0    6  80   0 - 529118 -      22:23 pts/0    00:00:00 ./s
1 R kali        5442    1136    5518  0    5  80   0 - 527069 -      22:23 pts/0    00:00:00 ./s
El thread que ejecuta revision de columnas es: 5442
La solucion al sudoku SI es valida
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S kali        5442    1136    5442  0    1  80   0 - 516824 -      22:23 pts/0    00:00:00 ./s

┌──(kali㉿kali)-[~/Desktop/Labs/lab3]
└─$
```

Responda las siguientes preguntas:

1. ¿Qué es una *race condition* y por qué hay que evitarlas?
   Se llega a dar cuando varios threads empezando desde dos, pueden acceder a los datos compartidos e intentan cambiar la información de estos al mismo tiempo. Es necesario evitarlas porque puede causar mal funcionamiento de un programa. En ejemplo puede que el thread 1 realicé una acción y el resultado sea usado por otra acción, esa acción depende del resultado del thread 1. Pero viene el thread 2 y cambie el resultado entonces la acción consecuente será totalmente diferente a lo que resultaría del thread 1.
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?
   Esta relación consiste en que ambos crean nuevos subprocesos los cuales realizan lo que se les indique por lo tanto se relacionan en ese sentido. Su diferencia parte en que clone() crea un proceso basado en el proceso ya existente. Por otro lado, en la mayoría de casos es favorable utilizar Pthreads, debido a que al crear nuevos procesos utilizando clone()incluye mucho overhead agregado además de necesitar de un canal para poder comunicarse tales como son los sockets, archivos o memoria compartida.
3. ¿Dónde, en su programa, hay paralelización de tareas, y dónde de datos?
   En el programa hay paralelización de tareas al momento de utilizar threads ya que se ejecutan varias tareas que se reflejan en el proceso principal. En la revisión de columnas y filas se puede notar la paralelización debido a es que este proceso se realiza en ocasiones repetidas durante toda la ejecución.

4. Al agregar los `#pragma`s 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.



Al agregar los pragmas for's cuando se están validando las columnas hay 8 LWP's, mientras que cuando está terminando hay 4. Deben haber 8 y 4 user threads respectivamente por el modelo 1 a 1 de multithreading de Linux

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?

   En la ejecución y código ocurren 4 LWP que son los generados por OpenMP en el transcurso de la revisión de las columnas y los threads que se crean a partir del uso de OpenMP, son threads que se tienen a la disponibilidad dependiendo de la cantidad de procesadores. En este caso se le asignó 4 a la máquina virtual.

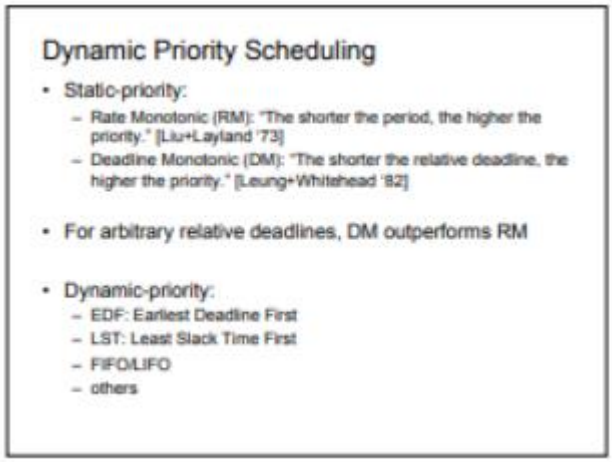
6. Observe cuáles LWP's están abiertos durante la revisión de columnas según `ps`. ¿Qué significa la primera columna de resultados de este comando? ¿Cuál es el LWP que está inactivo y por qué está inactivo? ***Hint***: consulte las páginas del manual sobre `ps`.

   Consultando la documentación, en la primera columna se indican los kernel threads y banderas del proceso. Revisando PS se nota que un thread se mantiene en un estado diferente a los demás durante el canal de wait(), esto indica que es un thread principal y se mantiene inactivo porque está bloqueado en estado fast space mutex.

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*?

   Es un grupo de threads donde todos son dirigidos por el master thread creado en líneas de pragma. Este mismo es el responsable de la revisión e imprime el texto indicado en el print dentro del programa, ya que los demás threads del grupo son copiados y reproducidos en un fork del master. En el tamaño del grupo se asigna una cantidad de threads ya configuradas, este ya indicado en el ejercicio del laboratorio. El término busy wait indica que el proceso chequea repetidamente la condición verificando si es True y también se utlizia para dejar agregado un tiempo de espera. OpenMP crea una cantidad indicada de threads y se crean en región paralélela para ser eliminados en un futuro en la última región.

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?
   Schedule(dynamic) de manera predeterminada funciona como FiFo y de esta manera distribuye el trabajo OpenMP.



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.
   Existe más ocurrencia ya que en la salida de PS despliega una mayor cantidad de LWP previstas en la revisión de las columas y al utilizar OpenMP se agrega Overhead ya que varios threads se comunican entre sí y es más lento (no en gran cantidad) porque el proceso es mutex y se realiza de esa manera para que no existan deadlocks incensarios, todo esto para seguridad del proceso.
10. ¿Cuál es el efecto de agregar `omp_set_nested(true)` ? Explique.
    Los números al momento de la revisión de columnas es diferente en los threads, comparado a cuando no se utilizó omp_set_nested. Anteriormente solo se podían observar set num threads la cual era la cantidad de threads desplegadas en PS que eran menores.

    Deja el max-active-levels-var al número activo de niveles de paralelismo que soporta la implementación. Cabe mencionar que en la documentación de OpenMP esta rutina se considera obsolete.