Universidad del Valle de Guatemala
Sistemas Operativos - Sección 10

Christopher Sandoval 13660
18/03/2020

Laboratorio 5

Primera ejecución:

```
oscreader@OSC:~/Desktop/Lab_5$ ./SudokuValidator El thread en el que se ejecuta la revision de columnas es: 28144
El thread en el que se ejecuta la revision de columnas es: 28144
El thread en el que se ejecuta la revision de columnas es: 28144
El thread en el que se ejecuta la revision de columnas es: 28144
El thread en el que se ejecuta la revision de columnas es: 28144
El thread en el que se ejecuta la revision de columnas es: 28144
El thread en el que se ejecuta la revision de columnas es: 28144
El thread en el que se ejecuta la revision de columnas es: 28144
El thread en el que se ejecuta la revision de columnas es: 28144
El thread en el que se ejecuta el main es: 28142
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+ 28142 26792 28142  0    1  80   0 -  2639 wait   02:27 pts/0    00:00:00 ./SudokuValidator

SUDOKU CORRECTO
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+ 28142 26792 28142  0    1  80   0 -  2639 wait   02:27 pts/0    00:00:00 ./SudokuValidator
```

- **¿Qué es una race condition y por qué hay que evitarlas?**
  Una race condition es una situación que puede ocurrir si hay dos procesos que quieren acceder o modificar una variable o archivo simultáneamente y como esto puede causar que el programa se ejecute erróneamente o entre a un deadlock si no se toman en cuenta.

- **¿Cuál es la relación, en Linux, entre pthreads y clone()? ¿Hay diferencia al crear threads con uno o con otro? ¿Qué es más recomendable?**
  La diferencia es que clone() crea un nuevo proceso que es una copia del proceso padre por lo que se copian todas las variables, código y estados del proceso original. A diferencia un pthread se encarga de ejecutar una función específica y no copia todo el código del proceso original. Depende el caso una puede ser mejor que la otra. La ventaja del clone() es que el proceso hijo es independiente por lo que si se traba lo podemos terminar sin afectar al proceso original. Un pthread tiene mucho menos overhead por no copiar todo el estado del proceso original pero si se traba es mas complicado acabarlo sin afectar al proceso original.

- **¿Dónde, en su programa, hay paralelización de tareas, y dónde de datos?**
  Hay paralelización en los fors que calculan las filas y las columnas del sudoku. La paralelización de datos hay cuando se accesa el array que contiene la matriz de datos que contiene el sudoku.

Con #pragma omp parallel for:

```
oscreader@OSC:~/Desktop/Lab_5$ ./SudokuValidator
El thread en el que se ejecuta la revision de columnas es: 28702
El thread en el que se ejecuta la revision de columnas es: 28702
El thread en el que se ejecuta la revision de columnas es: 28702
El thread en el que se ejecuta la revision de columnas es: 28704
El thread en el que se ejecuta la revision de columnas es: 28704
El thread en el que se ejecuta la revision de columnas es: 28703
El thread en el que se ejecuta la revision de columnas es: 28703
El thread en el que se ejecuta la revision de columnas es: 28705
El thread en el que se ejecuta la revision de columnas es: 28705
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+ 28700 26792 28700  0    5  80   0 -  9042 futex_ 02:43 pts/0    00:00:00 ./SudokuValidator
1 S oscread+ 28700 26792 28702  0    5  80   0 -  9042 futex_ 02:43 pts/0    00:00:00 ./SudokuValidator
1 R oscread+ 28700 26792 28703  0    5  80   0 -  9042 -      02:43 pts/0    00:00:00 ./SudokuValidator
1 R oscread+ 28700 26792 28704  0    5  80   0 -  9042 -      02:43 pts/0    00:00:00 ./SudokuValidator
1 R oscread+ 28700 26792 28705  0    5  80   0 -  9042 -      02:43 pts/0    00:00:00 ./SudokuValidator
El thread en el que se ejecuta el main es: 28700

SUDOKU CORRECTO
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+ 28700 26792 28700  0    4  80   0 -  9327 wait   02:43 pts/0    00:00:00 ./SudokuValidator
1 S oscread+ 28700 26792 28706  0    4  80   0 -  9327 futex_ 02:43 pts/0    00:00:00 ./SudokuValidator
1 S oscread+ 28700 26792 28707  0    4  80   0 -  9327 futex_ 02:43 pts/0    00:00:00 ./SudokuValidator
1 S oscread+ 28700 26792 28708  0    4  80   0 -  9327 futex_ 02:43 pts/0    00:00:00 ./SudokuValidator
```

- **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?**
  El main tiene 4 LWPs abiertos (por la revisión de filas) y la revisión de columnas tiene 4. El main crea cuatro threads cuando se revisan las filas y la revisión de columnas tiene abre cuatro threads a la hora de ejecutar los fors. Deben de haber 4 threads de usuario.

Con omp_set_num_thread(1):

```
oscreader@OSC:~/Desktop/Lab_5$ ./SudokuValidator
El thread en el que se ejecuta la revision de columnas es: 28802
El thread en el que se ejecuta la revision de columnas es: 28802
El thread en el que se ejecuta la revision de columnas es: 28802
El thread en el que se ejecuta la revision de columnas es: 28803
El thread en el que se ejecuta la revision de columnas es: 28803
El thread en el que se ejecuta la revision de columnas es: 28804
El thread en el que se ejecuta la revision de columnas es: 28804
El thread en el que se ejecuta la revision de columnas es: 28805
El thread en el que se ejecuta la revision de columnas es: 28805
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+ 28800 26792 28800  0    5  80   0 -  9578 futex_ 02:50 pts/0    00:00:00 ./SudokuValidator
1 S oscread+ 28800 26792 28802  0    5  80   0 -  9578 futex_ 02:50 pts/0    00:00:00 ./SudokuValidator
1 R oscread+ 28800 26792 28803  0    5  80   0 -  9578 -      02:50 pts/0    00:00:00 ./SudokuValidator
1 S oscread+ 28800 26792 28804  0    5  80   0 -  9578 futex_ 02:50 pts/0    00:00:00 ./SudokuValidator
1 R oscread+ 28800 26792 28805  0    5  80   0 -  9578 -      02:50 pts/0    00:00:00 ./SudokuValidator
El thread en el que se ejecuta el main es: 28800

SUDOKU CORRECTO
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+ 28800 26792 28800  0    1  80   0 -  9327 wait   02:50 pts/0    00:00:00 ./SudokuValidator
```

- **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 revisión de columnas tiene cinco LWPs abiertos. El main solo tiene un LWP abierto. OpenMP abre cuatro threads por defecto. Uno por cada núcleo del procesador asignado.

- **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?**
  El LWP que está inactivo es el primero, el que se creó en el main porque ese está en espera hasta que el cálculo de columnas termine.

- **¿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?**
  Un thread team es cuando un conjunto de threads están ejecutando en paralelo una misma tarea, en este caso cada caso del for. El master thread es el thread que esta a cargo de que el thread team complete su tarea. El master thread es el que parece que no hace nada pero es el que se encarga de asegurarse que los demás threads terminen su trabajo. Busy-wait se refiere a un estado en donde un proceso verifica constantemente que se haya completado una condición. OpenMP crea un thread pool a la hora que es llamado y crea los threads establecidos, los cuales son destruidos hasta que se termine de ejecutar el último.

Con schedule(dynamic):

```
oscreader@OSC:~/Desktop/Lab_5$ ./SudokuValidator
El thread en el que se ejecuta la revision de columnas es: 29926
El thread en el que se ejecuta la revision de columnas es: 29926
El thread en el que se ejecuta la revision de columnas es: 29926
El thread en el que se ejecuta la revision de columnas es: 29926
El thread en el que se ejecuta la revision de columnas es: 29926
El thread en el que se ejecuta la revision de columnas es: 29926
El thread en el que se ejecuta la revision de columnas es: 29923
El thread en el que se ejecuta la revision de columnas es: 29924
El thread en el que se ejecuta la revision de columnas es: 29925
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+ 29921 26792 29921  0    5  80   0 -  9042 futex_ 03:00 pts/0    00:00:00 ./SudokuValidator
1 R oscread+ 29921 26792 29923  0    5  80   0 -  9042 -      03:00 pts/0    00:00:00 ./SudokuValidator
1 R oscread+ 29921 26792 29924  0    5  80   0 -  9042 -      03:00 pts/0    00:00:00 ./SudokuValidator
1 R oscread+ 29921 26792 29925  0    5  80   0 -  9042 -      03:00 pts/0    00:00:00 ./SudokuValidator
1 R oscread+ 29921 26792 29926  0    5  80   0 -  9042 -      03:00 pts/0    00:00:00 ./SudokuValidator
El thread en el que se ejecuta el main es: 29921

SUDOKU CORRECTO
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+ 29921 26792 29921  0    1  80   0 -  9327 wait   03:00 pts/0    00:00:00 ./SudokuValidator
```

- **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 son nueve ya que al setearlo como dinámico se determina la cantidad de los threads según la necesidad. La distribución por defecto que hace OpenMP es estática y solo crea un número predeterminado de threads.

Con omp_set_num_threads(9):

```
oscreader@OSC:~/Desktop/Lab_5$ ./SudokuValidator
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
El thread en el que se ejecuta la revision de columnas es: 1723
El thread en el que se ejecuta la revision de columnas es: 1723
El thread en el que se ejecuta la revision de columnas es: 1731
0 S oscread+  1721  1499  1721  0   10  80   0 - 19286 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
El thread en el que se ejecuta la revision de columnas es: 1729
El thread en el que se ejecuta la revision de columnas es: 1727
1 S oscread+  1721  1499  1723  0   10  80   0 - 19287 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  1721  1499  1724  0   10  80   0 - 19287 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
El thread en el que se ejecuta la revision de columnas es: 1728
El thread en el que se ejecuta la revision de columnas es: 1726
El thread en el que se ejecuta la revision de columnas es: 1724
El thread en el que se ejecuta la revision de columnas es: 1730
1 R oscread+  1721  1499  1725  0   10  80   0 - 19287 -      01:49 pts/0    00:00:00 ./SudokuValidator
El thread en el que se ejecuta el main es: 1721
1 R oscread+  1721  1499  1726  0    3  80   0 - 13425 -      01:49 pts/0    00:00:00 ./SudokuValidator

SUDOKU CORRECTO
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+  1721  1499  1721  0    9  80   0 - 17523 wait   01:49 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  1721  1499  1732  0    9  80   0 - 17523 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  1721  1499  1733  0    9  80   0 - 17523 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  1721  1499  1734  0    9  80   0 - 17523 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  1721  1499  1735  0    9  80   0 - 17523 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  1721  1499  1736  0    9  80   0 - 17523 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  1721  1499  1737  0    9  80   0 - 17523 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  1721  1499  1738  0    9  80   0 - 17523 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  1721  1499  1739  0    9  80   0 - 17523 futex_ 01:49 pts/0    00:00:00 ./SudokuValidator
```

- **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?**

  Hay una mayor cantidad de concurrencia en el programa porque se crean más threads. No necesariamente hay un mejor desempeño ya que el procesador solo tiene cuatro threads. Separarlo en tantos threads puede causar que el desempeño empeore ya que habrá mucho más overhead que podría ser evitado separándolo en menos threads.

Con omp_set_nested(true):

```
oscreader@OSC:~/Desktop/Lab_5$ ./SudokuValidator
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+  2514  1499  2514  0   10  80   0 - 19286 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2516  0   10  80   0 - 19286 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 R oscread+  2514  1499  2517  0   10  80   0 - 19286 -      01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2518  0   10  80   0 - 19286 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 R oscread+  2514  1499  2519  0   10  80   0 - 19286 -      01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2520  0   10  80   0 - 19286 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2521  0   10  80   0 - 19286 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2522  0   10  80   0 - 19286 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2523  0   10  80   0 - 19286 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2524  0   10  80   0 - 19286 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
El thread en el que se ejecuta la revision de columnas es: 2517
El thread en el que se ejecuta la revision de columnas es: 2516
El thread en el que se ejecuta la revision de columnas es: 2523
El thread en el que se ejecuta la revision de columnas es: 2521
El thread en el que se ejecuta la revision de columnas es: 2519
El thread en el que se ejecuta la revision de columnas es: 2518
El thread en el que se ejecuta la revision de columnas es: 2524
El thread en el que se ejecuta la revision de columnas es: 2522
El thread en el que se ejecuta la revision de columnas es: 2520
El thread en el que se ejecuta el main es: 2514

SUDOKU CORRECTO
F S UID        PID  PPID   LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S oscread+  2514  1499  2514  0    9  80   0 - 23670 wait   01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2525  0    9  80   0 - 23670 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2526  0    9  80   0 - 23670 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2527  0    9  80   0 - 23670 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2528  0    9  80   0 - 23670 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2529  0    9  80   0 - 23670 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2530  0    9  80   0 - 23670 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2531  0    9  80   0 - 23670 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
1 S oscread+  2514  1499  2532  0    9  80   0 - 23670 futex_ 01:54 pts/0    00:00:00 ./SudokuValidator
```

- **¿Cuál es el efecto de agregar omp_set_nested(true)?**
  Lo que hace esta instrucción es que un thread pueda crear más threads si es posible para completar su ejecución.