Universidad del Valle de Guatemala
José Gabriel Block Staackmann

HDT3:

1. ¿Qué es una race condition y por qué hay que evitarlas?
   Es cuando el comportamiento de un sistema intenta hacer dos acciones diferentes al mismo tiempo. Hay que evitar esto para poder tener bajo control todo comportamiento del sistema, para evitar que sea impredecible.

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?
   La función clone() es una llamada al sistema que puede crear threads, y procesos. Estas funciones se diferencian en que una puede crear procesos y cambian los parámetros, por lo que clone es una función más compleja, que requiere de más parámetros que pthread_create(). Por lo tanto, si se da el caso de uso de crear threads, conviene más pthread_create().

3. ¿Dónde, en su programa, hay paralelización de tareas, y dónde de datos?
   En los forks del main, que reparten las peticiónes de revisión de filas, columnas y cuadros es paralelismo de tareas, ya que son diferentes tareas con la misma base de datos. En el caso de paralelismo de datos se da en las tareas de revisión, que se tratan los datos de diferente forma, pero la tarea es la misma, revisar que los números sean de 1 a 9 y no se repitan.

4. Al agregar los #pragmas a los ciclos for, ¿cuántos LWP'shay 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.
   En el progreso de revisión de columnas hay 5 LWP abiertos. Y antes de terminar el main hay 4 LWP por lo tanto serían 4 user threads porque Linux trabaja con el modelo de multithreading uno a uno.

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?
   Hay solo 1 LWP en ambos, o sea que OpenMP crea un thread 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 es el estado de actividad por thread, si es 1 esta activo y si es 0 esta inactivo. En mi caso, solo 1 está inactivo, esto quiere decir que ese es el thread master y que los que están activos son los slaves.

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

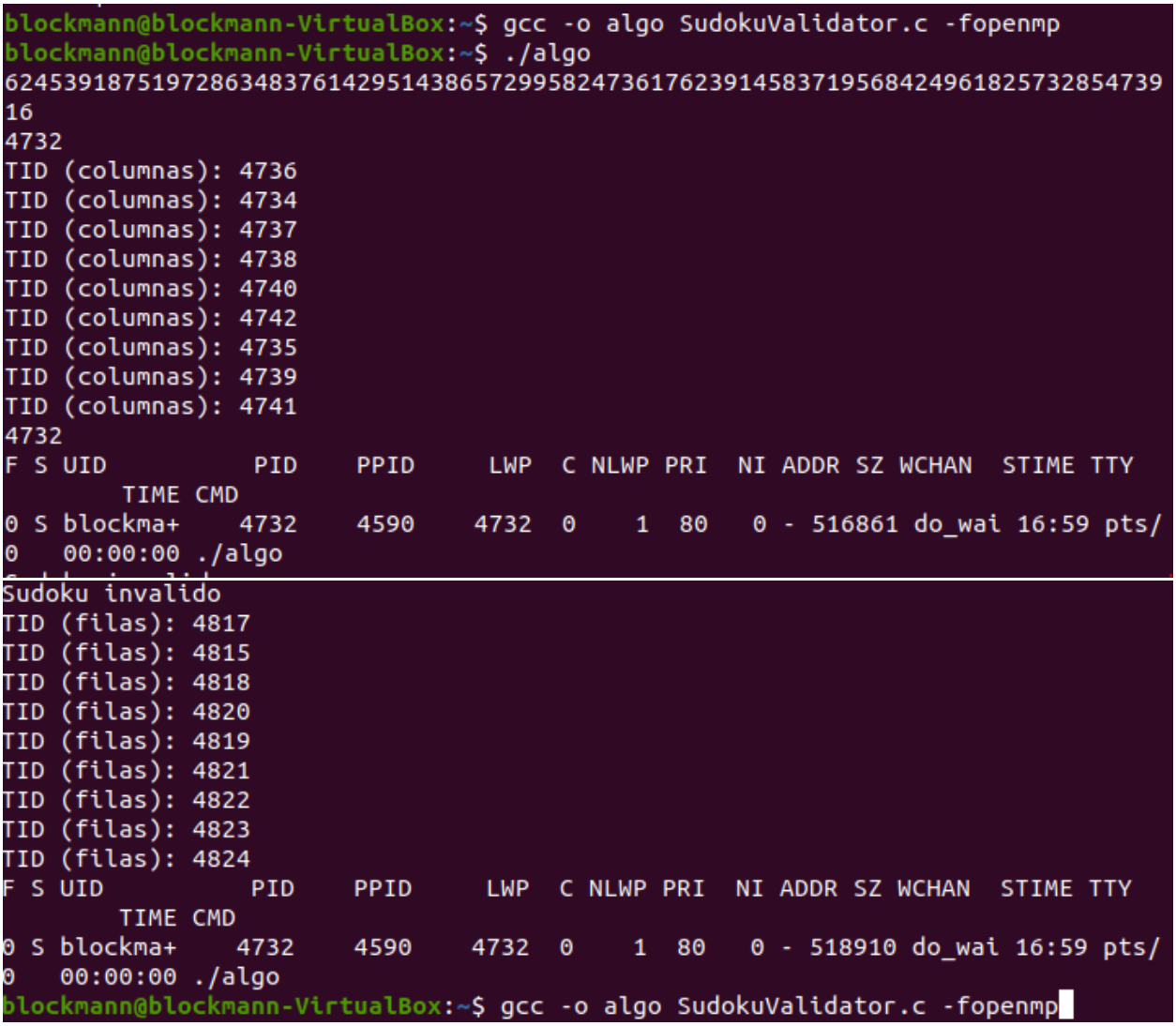
en este caso? ¿Por qué parece haber un thread “corriendo”, pero que no está haciendo nada? ¿Qué significa el término busy-wait? ¿Cómo maneja OpenMP su thread pool?
Un thread team es un conjunto de threads que crea OpenMP en el cual el thread que esta inactivo es el master y los activos los slaves.
Busy-wait hace referencia a una técnica en la que se hace una espera hasta que se ocurra algun efecto esperado. Como un semáforo.
Este maneja el thread pool de manera que si hay tareas, los threads de la pool van a hacer las tareas y en caso de ser insuficientes, se crean más threads en la pool para poder completar las tareas.

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?
   Se crearon la misma cantidad de LWP para cada forma, en mi caso fueron 9 por forma. Por lo tanto, podemos comprobar y afirmar que OpenMP hace un buen trabajo con la distribución de tareas.

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 menos concurrencia sin no se ha agregado omp_set_nested(true), esto no quiere decir que sea más eficiente, ya que se crean menos threads.

```
blockmann@blockmann-VirtualBox:~$ gcc -o algo SudokuValidator.c -fopenmp
blockmann@blockmann-VirtualBox:~$ ./algo
624539187519728634837614295143865729958247361762391458371956842496182573285473916
5253
TID (columnas): 5258
TID (columnas): 5257
TID (columnas): 5255
TID (columnas): 5255
TID (columnas): 5255
TID (columnas): 5257
TID (columnas): 5258
TID (columnas): 5256
TID (columnas): 5256
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY
        TIME CMD
0 S blockma+    5253    4590    5253  0    5  80   0 - 74466 futex_ 20:16 pts/0
    00:00:00 ./algo
1 R blockma+    5253    4590    5255  0    5  80   0 - 74466 -      20:16 pts/0
    00:00:00 ./algo
1 R blockma+    5253    4590    5256  0    5  80   0 - 74466 -      20:16 pts/0
    00:00:00 ./algo
1 R blockma+    5253    4590    5257  0    5  80   0 - 74466 -      20:16 pts/0
    00:00:00 ./algo
1 R blockma+    5253    4590    5258  0    5  80   0 - 74466 -      20:16 pts/0
    00:00:00 ./algo
5253
Sudoku invalido
TID (filas): 5263
TID (filas): 5263
TID (filas): 5263
TID (filas): 5263
TID (filas): 5263
TID (filas): 5263
TID (filas): 5263
TID (filas): 5262
TID (filas): 5261
F S UID          PID    PPID     LWP  C NLWP PRI  NI ADDR SZ WCHAN  STIME TTY
        TIME CMD
0 S blockma+    5253    4590    5253  0    5  80   0 - 74493 do_wai 20:16 pts/0
    00:00:00 ./algo
1 R blockma+    5253    4590    5259  0    4  80   0 - 74493 -      20:16 pts/0
    00:00:00 ./algo
blockmann@blockmann-VirtualBox:~$
```

10. ¿Cuál es el efecto de agregar omp_set_nested(true)? Explique
    Se crean muchos más threads en los thread teams porque se da el nested paralelism.


blockmann@blockmann-VirtualBox:~$ gcc -o algo SudokuValidator.c -fopenmp
blockmann@blockmann-VirtualBox:~$ ./algo
624539187519728634837614295143865729958247361762391458371956842496182573285473916
4732
TID (columnas): 4736
TID (columnas): 4734
TID (columnas): 4737
TID (columnas): 4738
TID (columnas): 4740
TID (columnas): 4742
TID (columnas): 4735
TID (columnas): 4739
TID (columnas): 4741
4732
F S UID PID PPID LWP C NLWP PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
0 S blockma+ 4732 4590 4732 0 1 80 0 - 516861 do_wai 16:59 pts/0 00:00:00 ./algo
Sudoku invalido
TID (filas): 4817
TID (filas): 4815
TID (filas): 4818
TID (filas): 4820
TID (filas): 4819
TID (filas): 4821
TID (filas): 4822
TID (filas): 4823
TID (filas): 4824
F S UID PID PPID LWP C NLWP PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
0 S blockma+ 4732 4590 4732 0 1 80 0 - 518910 do_wai 16:59 pts/0 00:00:00 ./algo
blockmann@blockmann-VirtualBox:~$ gcc -o algo SudokuValidator.c -fopenmp