Universidad del Valle de Guatemala
Sistemas Operativos
Docentes: Erick Pineda; Tomás Gálvez P.
Semestre I, 2021

Laboratorio #3

Fecha de Entrega: 03 de marzo, 2021.

Estudiante: **Estuardo Ureta 17010**

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.

Responda las siguientes preguntas:

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

Se puede definir como un comportamiento extraño debido a una dependencia crítica inesperada del momento relativo de los eventos o procesos. Las race conditions generalmente involucran a uno o más procesos que acceden a un recurso compartido (como un archivo o una variable), donde este acceso múltiple no se ha controlado adecuadamente.
Hay que evitarlas para asegurarnos que si un proceso está utilizando una variable o archivo compartido, los otros procesos se excluirán de hacer las mismas cosas.

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?

Pthread se utiliza para multithreading. La principal diferencia entre procesos y threads es que los threads comparten un único espacio de memoria, mientras que los procesos tienen cada uno su propio espacio de memoria. Clone es una llamada al sistema de bajo nivel específica de Linux y se puede usar para crear estos procesos e hilos.
Siempre que nuestro código esté limpio, se puede ganar algo de eficiencia utilizando varios pthreads en un solo proceso. El uso de varios procesos agrega un poco de overhead, pero puede hacer que nuestro código sea un poco más robusto, porque limita el daño que puede causar un solo problema y hace que sea mucho más fácil cerrar y reemplazar un proceso si se encuentra con un problema importante.

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

El propósito de los fors, en conjunto con OpenMP, es general paralelismo. Entonces, cuando ejecutamos en distintas nidos estos fors estamos haciendo paralelización de tareas ya que son las misma tareas con distintos datos. Mientras que la paralelización de datos se da con los mismos datos pero distintas funciones (fila, columna, 3x3).

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 abierto en cada caso, entonces? Hint: recuerde el modelo de multithreading qué usanLinuxy Windows.

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?

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.

7. Comparelos resultados de psen la pregunta anterior con los que son desplegados por la función de revisión de columnas per se.¿Qué es un thread teamen 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?

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?

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.

10. ¿Cuál es el efecto de agregar omp_set_nested (true)? Explique

Esta variable de entorno habilita o deshabilita el paralelismo anidado. Si el paralelismo anidado está deshabilitado, las regiones paralelas anidadas se serializan y se ejecutan en el thread en efecto. Las regiones paralelas anidadas pueden emplear threads adicionales según estén disponibles.

NO ME CORRIO EL PROGRAMA POR LO QUE NO PUDE CONTESTAR TODAS LAS PREGUNTAS