Universidad del Valle de Guatemala Yasmin Chávez

Sistemas Operativos Carné: 16101

Laboratorio #5

- **¿Qué es una race condition y por qué hay que evitarlas?**
  Sucede cuando 2 o más threads pueden acceder a memoria compartida y quieren cambiarla al mismo tiempo. Es importante evitarlas para que no suceda sobreescritura o lecturas de valores incorrectos, según lo que estemos programando.
- **¿Cuál es la relación entre pthreads y clone()? ¿Hay diferencia al crear threads con uno o con otro? ¿Qué es más recomendable?**
  Con ambos podemos crear procesos. Al usar clone también se necesitan tener canales de comunicación. La ventaja que provee es que si un proceso hijo falla podemos matarlo y comenzar con una nuevo de forma fácil. Con threads se debe tener cuidado con los accesos a la memoria, es importante las implementaciones de lock's, pero su uso pueden dar mayor eficiencia al programa.
- **¿Dónde, en su programa, hay paralelización de tareas, y dónde de datos?**

  Hay paralelización de tareas al momento que en nuestro programa utilizamos OpenMP, nos permite añadir concurrencia en nuestros ciclos for. Se utiliza paralelismo de datos cuando usamos threads a la revisión de filas y otros a la de columnas.

- **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? Hint: recuerde el modelo de multithreading que usa Linux.**
  Antes de terminar el main deben haber abiertos los LWP´s que faltan por terminar sus tareas, durante la revisión de columnas deben haber 9 threads. En cada caso debe haber un user thread abierto por cada uno.
- **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 crea OpenMP (en general) por defecto?**
  Solo hay uno, en este caso no se estaría aprovechando el potencial de paralelismo que tiene el programa. Anteriormente habían más LWP´s abiertos. Genera 8 threads por defecto.

- **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?**
  Hay 9 LWP´s abiertos durante la revisión de columnas. La primera columna es el identificador del proceso. Se encuentra inactivo porque no se esta realizando la tarea en ese momento.
- **Compare los resultados de ps en la pregunta anterior con los que son desplegados por el método de revisión de columnas perse. ¿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?**
  Master thread es como el thread principal, el original y tiene ID 0, ese thread es el que nos da paso al busy-waiting que es como tener un ciclo infinito corriendo. OpenMP se usa para paralelismo dentro de un nodo (multi-core).
- **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 el método 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?**
  La distribución de trabajo en OpenMp por defecto es la creación de threads por cada proceso que permita el paralelismo. En LWP's se puede reducir la metadata del proceso.
- **Luego de agregar las llamadas omp_set_num_threads() a cada método donde se usa OpenMP, y luego de ejecutar su programa con y sin la cláusula schedule() en cada for, ¿hay más o menos concurrencia en su programa? ¿Es esto sinónimo de un mejor desempeño? Explique.**
  El tamaño por defecto se puede inicializar con la variable omp_num_threads. Durante la ejecución se configura un total de hilos correspondiente con el total de procesadores disponibles. Mientras que si usamos la función omp_set_num_threads() nosotros definimos los hilos correspondientes, entonces queda en nuestras manos la eficiencia porque podemos asignar hilos extra o talvez asignamos muy pocos y se podría mejorar la eficiencia del programa.
- **¿Cuál es el efecto de agregar omp_set_nested(true)? Explique.**

  Permite añadir regiones paralelas anidadas. Si esta True entonces los miembros de un equipo de hilos pueden crear nuevos equipos de hilos.